# THERMAL MODELING OF MANY-CORE PROCESSORS

A Thesis
Presented to
The Academic Faculty

by

Nikhil Sathe

In Partial Fulfillment
of Requirements for the Degree
Master of Science in
School of Electrical and Computer Engineering

Georgia Institute of Technology
August 2010

COPYRIGHT © 2010 BY NIKHIL SATHE

# THERMAL MODELING OF MANY-CORE PROCESSORS

Approved by:

Dr. Saibal Mukhopadhyay, Advisor
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Dr. Sudhakar Yalamanchili
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Dr. Satish Kumar
School of Mechanical Engineering
*Georgia Institute of Technology*

Date Approved: 29th June 2010

# ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Saibal Mukhopadhyay for his supervision, advice and guidance throughout the duration of this work. Above all, he provided me unflagging encouragement and support in various ways. His constant enthusiasm for this project was truly motivating. I attribute the level of my Masters degree to his support without which this work would not have been possible.

I also sincerely appreciate the guidance of Dr. Sudhakar Yalamanchili and Dr. Satish Kumar. This research work benefited significantly from their inputs. I would also like to thank them for being a part of my thesis reading committee.

I would like to thank Dr. Rahul Rao (IBM) and Dr. Arijit Raychowdhury (Intel Corp.) for their valuable inputs. I would like to thank Intel Corporation and IBM for providing the financial support for this research project.

I gratefully acknowledge my colleagues Minki Cho and William Song for their technical support and crucial contributions towards the completing of this work. I would also like to acknowledge Subho Chatterjee and Jeremy Tolbert for all the technical and non-technical interactions in the GREEN lab. I thank all my friends for sharing my special moments and making this journey exciting. I specially thank Amruta Varudkar for sharing this journey with me.

None of this would have ever been possible without the unwavering support and blessing of my parents and brother.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF SYMBOLS AND ABBREVIATIONS

| Abbreviation/ Symbol | Meaning |
|---|---|
| CMOS | Complementary metal oxide semiconductor |
| $V_t$ | Threshold voltage |
| GUI | Graphical user interface |
| IC | Integrated circuit |
| PMOS | P-type metal oxide semiconductor |
| NMOS | N-type metal oxide semiconductor |
| A | Weight parameter |

## SUMMARY

Future generation processors will rely on multiple on-chip cores to sustain exponential growth in the performance requirement. Such processors with multiple cores are expected to be power hungry. The high power dissipation in such processors will lead to high operating temperatures. Intelligent thermal management techniques are required to manage the thermal field in such future generation processors.

If the number of cores in the processor increases, it presents unique challenges from the thermal perspective. These challenges need to be understood and addressed during the design phase and also at run-time. This work aims to model, analyze and understand the thermal behavior of such future generation manycore processors.

We develop an infrastructure with a rich feature set in order to simulate steady state and transient thermal behavior of manycore processors considering in detail the physical properties of heat flow, thermal management techniques and thermal sensor properties. The developed infrastructure is based on a thermal simulator called HotSpot, and is used to explore many-core thermal management techniques.

Our aim is to propagate this environment to the computer architecture curriculum. This has led us to develop a graphical user interface based modeling tool which can be used by students in order to simulate the thermal behaviors of manycore processors and gain valuable insights to the same.

# 1 INTRODUCTION

## 1.1 Moore's Law

Over the last four decades, CMOS technology has been advancing rapidly. Transistor sizes have reduced over each generation making the transistors smaller and faster. In 1965, Gordon Moore predicted that the number of transistors would increase two times every two years. Another interpretation of this statement was that the performance would double every two years. This trend in CMOS technology is widely known as Moore's Law [29] and the semiconductor industry has been following it for more than four decades.

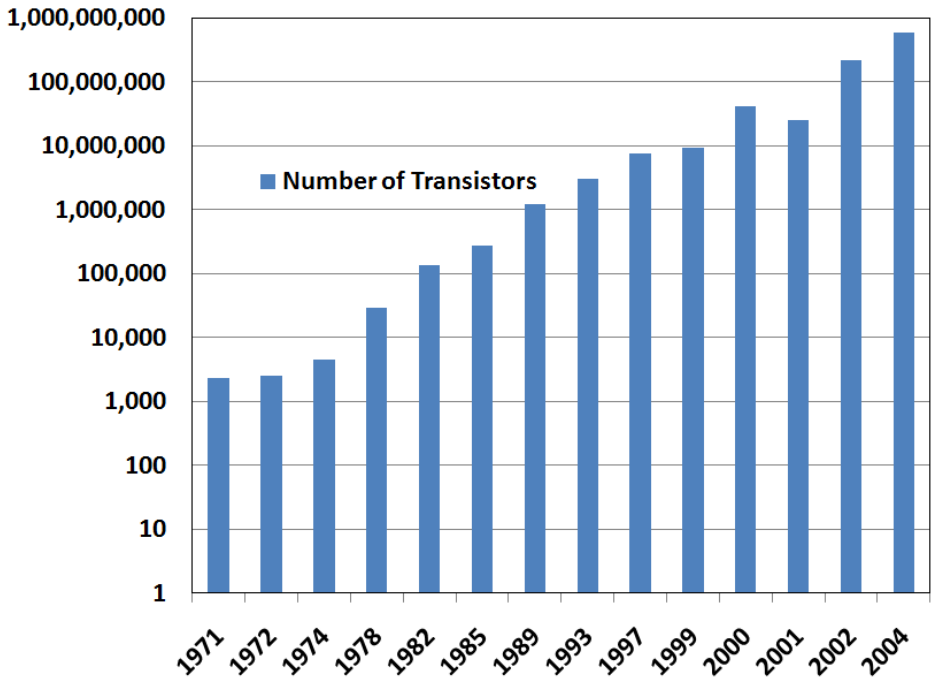



**Figure 1-1: Evolution of number of transistors per die [29]**

## 1.2 Power in CMOS circuits:

A direct outcome of aggressive technology scaling is the higher number of transistors integrated on a single die. This significant increase in the number of transistors per die has eventually led to significant increase in the on-die transistor density. This

exponential increase in the transistor density has increased the per-die power consumption of such integrated circuits (ICs). As the transistors have become faster over the generations, the operating frequency has also increased significantly. To understand the implication of higher transistor density and higher operating frequency on the power consumed by the ICs, we will delve deeper into the different components of power in CMOS circuits.

## 1.3 Basic CMOS power components

### Dynamic power:

In CMOS circuits, the power used to switch the logical state of the output is called the dynamic power. The dynamic power is necessary in order to do the required computation by the circuit. The dynamic power in CMOS circuits depends on multiple factors. The basic equation of dynamic power is given as follows:

$$P_{dynamic} = C_{load} \times {V_{DD}}^2 \times F$$

where,

$P_{dynamic}$ = dynamic power consumed by the circuit

$C_{load}$ = total output capacitance (this increases with the number of transistors in the IC)

$V_{DD}$ = supply voltage

$F$ = frequency of operation

From the equation, we can clearly see that the dynamic power depends on the total number of transistors in the IC and the frequency of operation. As a result, the dynamic power of the ICs has also gone up significantly over the generations.

### Static power:

The main component of the static power in CMOS is the sub-threshold leakage power. Aggressive technology scaling has caused sub-threshold leakage to increase due to the short channel effect [5]. This component is more important from the thermal

perspective because the sub-threshold leakage depends exponentially on the operating temperature. The general equation for the sub-threshold current is given as:

$$I_D = I_o \, e^{\frac{q\,(V_{gs}-V_{th})}{nkT}} \left(1 - \, e^{-\frac{qV_{ds}}{kT}}\right)$$

Where,

$I_D$ = drain current,

$I_o$ = drain current at Vgs = Vth,

$V_{ds}$ = drain to source voltage,

$V_{th}$ = threshold voltage of the device,

$V_{gs}$ = gate to source voltage,

$T$ = operating temperature

As the leakage power is exponentially dependent on the operating temperature, it forms a positive feedback loop with temperature. As the temperature increases, so does the leakage power. This leakage power in turn increases the operating temperature. If not managed properly, this may lead to thermal runaway conditions. This makes the leakage power temperature interaction a very important component of the thermal analysis.

Another component of the static CMOS power is the gate leakage. As the channel length of the transistor is reduced, the thickness of the gate oxide has also reduced simultaneously to balance the vertical and horizontal electric fields in the transistor. This has caused the gate thickness of the transistor to become extremely small (around 3-5 atoms thick). This small gate thickness enables electrons to directly punch-through the very thin gate, and the phenomenon is called “Gate Tunneling”. The gate leakage power can be reduced using innovative technology solutions such as using Hi-K dielectric materials for the gate of the CMOS transistor.

## 1.4 Problems due to high power dissipation

In order to keep the throughput increasing over the generations, we have relied on technology scaling, frequency scaling and increasing number of transistors per die. This evolution has caused the power density in modern ICs to increase exponentially as shown in Figure 1-2.

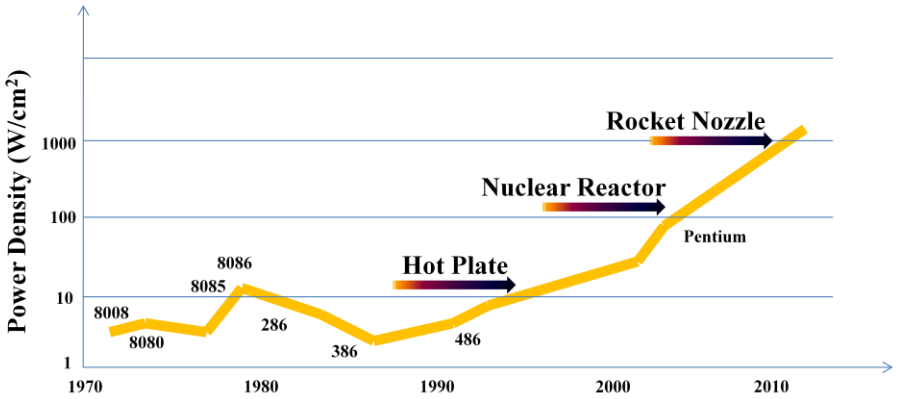



**Figure 1-2: Power density evolution [49]**

The significant increase in the absolute power and the on-die power density has rendered the conventional air cooling solutions incapable to operate effectively, causing operating temperatures to rise. If the CMOS devices are not cooled sufficiently, they face the possibility of permanent damage due to excessive electrical degradation. Therefore, while designing CMOS ICs today, we not only need to pay attention to performance but also to the power consumed by the circuits and the on-chip temperature. In summary, high operating temperature causes several problems in CMOS systems. Some of the problems can be elaborated as follows:

1. Higher operating temperature leads to a higher leakage power. Higher leakage power will in turn increase temperature of the IC. Thus, a positive feedback loop exists between leakage power and temperature. If the temperature (and leakage power) of the IC is not controlled, then the positive feedback loop causes seriously high temperatures in the IC which may cause thermal runaway conditions causing complete destruction of the devices.

2. Higher operating temperatures are harmful to the reliability of the devices and on-chip interconnects. Many of the degradation mechanisms in the IC such as electro-migration, stress migration and dielectric breakdown, etc are temperature dependent and are accelerated at elevated temperatures [37].

3. Higher operating temperature causes the transistors and interconnects to operate slower. These slower devices and interconnects may cause functional failures or degraded performance.

4. If the temperature of the IC is higher, more energy is needed to cool down the chip. This causes the cooling solutions to spend more energy in cooling the chip [22]. Moreover, a highly non-uniform thermal surface also degrades the package reliability [22]. Thus, we not only desire a low operating temperature, but also a uniform thermal profile if possible.

   .

## 1.5 Process variation

Extremely small transistor dimensions have also caused another problem. The manufacturing process used to manufacture the transistors on the silicon wafer has not scaled as fast as the transistor sizes. As a result, the accuracy with which the transistor (the channel width, the channel length and the gate thickness) can be manufactured has reduced [151, 152, 2]. This causes a variation in the physical parameters of the manufactured transistors and called “process variation”. As a result of process variation, two transistors which were designed to be same can behave very differently. A change in the physical parameters of the transistors due to process variation causes the threshold voltage of the devices to change. Sub-threshold leakage current depends on the threshold voltage - lower the threshold voltage, more the sub-threshold leakage current for the same temperature.

If we consider two dies which have the same design, the leakage current in these tow dies may be very different due to process variation. Such variation is called as inter-

die variation and is observed across different dies from same or different wafers. It is also possible that within the die, two transistors having same design have different leakage characteristics. Such variation within the die is called intra-die process variation [51].

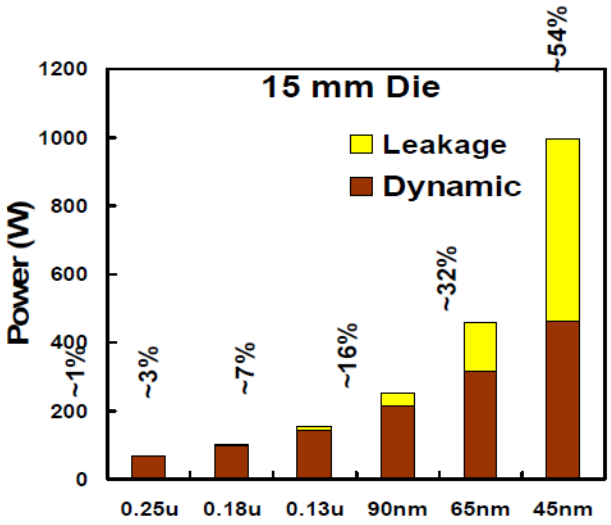



Figure 1-3: Power distribution in CMOS [Source: Intel]

Over the generations, we observe that the leakage power has become a significant portion of the total power dissipated by CMOS circuits. This trend is illustrated in Figure 1-3. Moreover, the leakage power in a given generation further increases for the low $V_t$ process corner. Thus, leakage power has become a dominant portion of the total power in highly scaled technologies which we are being used currently to manufacture semiconductor ICs, and has to be considered during modeling the thermal behavior of modern day ICs.

## 1.6 Evolution of multicore and manycore processors

In order to sustain the performance demand put forward by Moore's law, we need to keep on increasing the throughput of the processors. Increasing operating frequency of the processors is not an option since it will increase the dynamic power the IC thus increasing the power density. So, we need to explore other options of increasing throughput. To address this challenge, integration of a large number of on-chip cores has

emerged as a key performance scaling technique. This is referred to as many-core processors.

Another solution which has emerged recently is the use of asymmetric processors. In conventional multicore or manycore processors, all the cores are identical. Such processors in which all the cores are identical are called 'symmetric manycore processors'. If the processors have both, simple and complex cores, then we refer to such processors as 'asymmetric manycore processors'. The simple cores in the heterogeneous multiprocessors can take care of the thread level parallelism while the complex cores can deliver high performance when the applications are single threaded or have low thread level parallelism.

## 1.7 Thermal behavior of multicore and manycore processors

In case of manycore processors, the number of cores is relatively high. As a result, each core can potentially act as a hotspot in the IC, producing multiple hotspots. Moreover, the power profile of each of these threads is not deterministic. Therefore, the location of the hotspots will vary significantly over space and time. Figure 1-4 shows the variation of hotspots over space and time for a 256 core chip. The simulation was carried out to simulate a scenario where the number of active cores is around 25% of the total cores and the cores were turned on and off randomly to simulate a random thread scheduling algorithm.

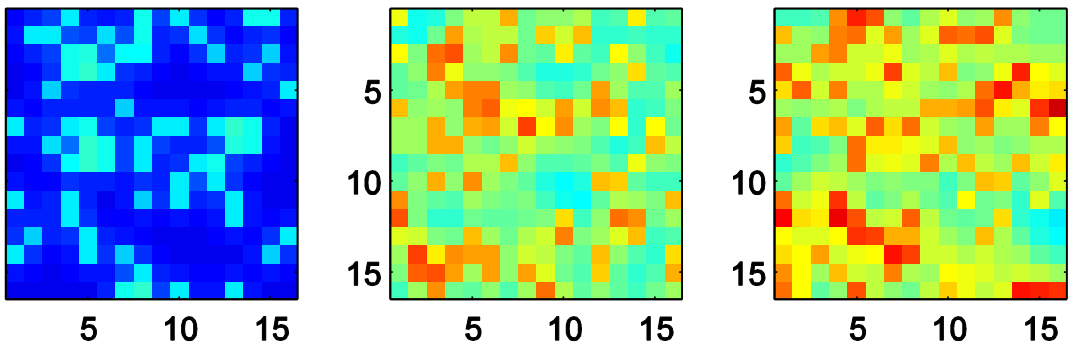



**Figure 1-4: Thermal profile of a manycore chip over time**

It is observed that the location of the hotspots keeps on changing with respect to time and space. As the number of cores increases, the variation of temperature on the chip also increases, producing more number of hotspots. As manycore chips have large number of simple cores, the dimensions of the cores are small. These small dimensions may cause thermal field in one core to significantly affect the temperature of the adjacent core. This introduces one more variable in the analysis on thermal fields in many-core chips.

## 1.8 Thermal management overview

The number of on-chip cores in increasing in order to sustain the increase in required performance. This has caused significant increase in the power of the chip. This has resulted in high operating temperatures. Control mechanisms are required in such cases to reduce the risk of thermal damage or degradation of the devices. In the worst case, the extremely high temperatures can cause permanent damage to the devices causing functional failures. Even if the devices are not damaged, other reliability problems arise as hotspots can accelerate failure mechanisms such as electro-migration, stress migration and dielectric breakdown [37]. For example, in [38], the authors show that a $10^{o}$ change in the operating temperature can cause the effective lifetime of the chip to change by 2x. Another important factor in thermal management is the uniformity of the thermal surface. In [22], it is shown that the cooling efficiency depends on the uniformity of the thermal surface. The more uniform is the thermal surface, less is the cooling energy required. The non-uniform thermal surface also poses challenges to the package reliability caused due plastic deformation and package fatigue.

In order to mitigate the various problems caused due to high operating temperatures, many mechanisms have been proposed over the years to control the on-chip temperature. These control mechanisms can be implemented in either software or hardware and can either be static or run-time. Using run-time solutions for controlling the on-chip temperature is more practical as they allow a higher performance for the same peak temperature. The different thermal control mechanisms can be classified into two groups depending on the control strategy they use [15]. In one case, the temperature is

controlled by reducing the power dissipated by the chip. The other group of mechanisms controls the chip temperature by distributing the activity over the entire chip area.

The simplest technique which changes the power dissipation and controls the temperature is the stop-and-go mechanism [15]. It is obvious that the stop-and-go mechanism stops the core as the temperature reaches a certain threshold value and controls the maximum chip temperature at the expense of performance. In [33], the authors describe a technique called advanced configuration and power interface. This technique activates of deactivates the input-output devices depending on the operating temperature of the chip. Other techniques discusses in [34] and [36] include a combination of using thermal sensors and then generating interrupts. These interrupts in turn inform the software of the high chip temperature. The system can then take corrective action for the generated interrupts (such as reducing the fetch bandwidth of the caches). This results in less core activity reducing the dynamic power of the chip and the chip temperature. In [35], the authors propose a technique which uses a feedback mechanism to inform the operating system about the chip temperature. This information is then used by the operating system to reduce the switching activity of the core. Dynamic task management (DTM) for controlling the on-chip temperature [14] can also use techniques such as decode throttling (which limits the instructions fed to the pipeline), speculation control (which limits the number of predicted by unresolved branches in the pipeline) and I-cache toggling. Other techniques such as dynamic power management [34] use methods such as clock gating, using qualified system latches (which enable clock gating at a finer granularity) and selective activation (where sub-modules are activated only if they are required). Another technique implemented to reduce the maximum on-chip temperature is to reduce the dynamic power of the core by reducing the operating frequency and supply voltage of a core. This technique is called Dynamic Voltage or Frequency Scaling [32]. All these techniques fall into the first category presented in [15]. As these management techniques rely on reducing the power dissipation to reduce the temperature, they suffer from performance degradation.

In order to mitigate the performance degradation, other thermal control mechanisms have been proposed which try to distribute the power over the entire chip. Lim et al [43] propose adding extra hardware resources and then shift the computation to

the redundant resources if the temperature of the primary resource increases beyond a certain threshold. [44] also relies on increasing hardware resources and then shift computation across the resources in case of thermal emergencies. Techniques which also exploit the inbuilt capability of the operating systems and hypervisor layers have also been proposed. The authors in [42] present an algorithm which tries to allocate threads (tasks) in an intelligent manner in order to reduce the chip temperature for e.g. the new threads are assigned to the coolest cores in the chip so that the overall temperature in the chip can be reduced. Thread scheduling techniques such as Heat-and-run [41] aim at exploiting simultaneous multi-threading and chip multi-processor characteristics to schedule threads having complementary resource requirements. In [41] the threads from a hot core are migrated to cooler cores. [45] extends this concept of thread scheduling and proposes OS level techniques which schedule the threads to the coolest possible cores in the chip. [45] also proposes the Adaptive-Random policy which modifies the scheduling algorithm based on the thermal history of the cores. [46] and [47] also propose different task scheduling techniques. Kursun et al [21] propose a technique called core hopping which balances the on-chip thermal profile by moving the computation from hotter cores to cooler cores. Several other techniques which use thread migration such as predictive dynamic thermal management [48] have been proposed, which predicts the future temperature and takes the necessary corrective action by migrating threads from future hottest cores to coolest cores.

## 1.9 Organization of the thesis

There is a need to manage the thermal field in manycore processors. This thesis presents an environment which includes many features which enable us to model different physical phenomenon and thermal management techniques required for modeling manycore processors. But the manycore processors pose several other unique challenges on top of the current challenges. Some of the previous works do address many of the problems of thermal management in manycore systems. Our aim is to come up with more such intelligent policies which include all the desirable features of the previous works. We also aim to couple thermal modeling infrastructure with an architecture

simulator which will then enable us to accurately estimate the performance implications of the different management policies right during the design phase. This will enable us to implement much more efficient techniques [125,126]. We can then compare the different policies for the target system and choose the optimal policy based on the results of this infrastructure.

The thesis is organized into 5 chapters. We introduce the importance of controlling the temperature in chapter 1. We then move on to explain the modeling environment (and the different features) that we have developed to understand the thermal behavior of manycore processors. In chapter 3, we discuss how the features of our modeling environment can be used to model the thermal field in manycore processors. Chapter 4 discusses the details of a thermal management policy which works on the notion of "thermal weight". Our aim is to percolate the knowledge gained through this research to other interested users who can then use the modeling infrastructure to analyze thermal properties of different manycore systems. We have built a simple GUI based modeling tool to make this possible. We give an overview of this GUI based tool in chapter 5.

# 2 THERMAL MODELING INFRASTRUCTURE

## 2.1 Structure of semiconductor ICs

Semiconductor ICs are produced by manufacturing transistors on a silicon wafer. This wafer is then cut into multiple dies. These dies are then placed in a package and then hermetically sealed. The package provides the ICs with mechanical strength and protection as well as the means of communicating with the outside world using pins. As we operate the IC, power is consumed by the transistors and heat is produced. Several thermal materials are placed between the devices and the outside world in order to make the heat transfer more efficient.

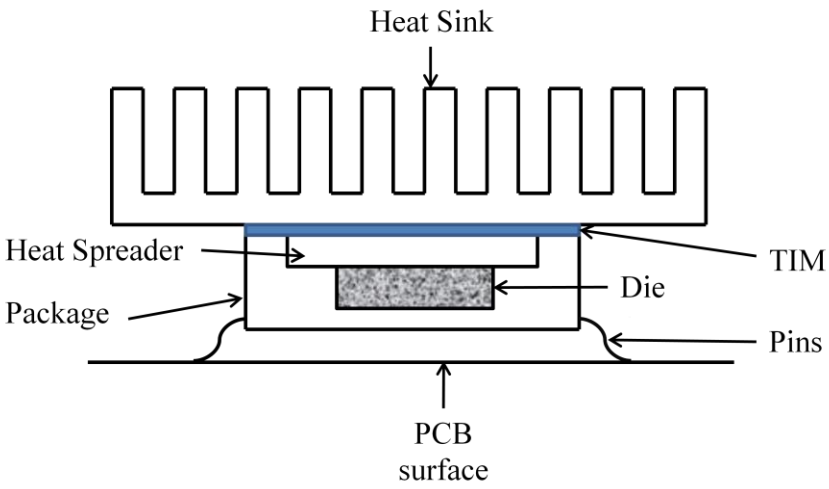



**Figure 2-1: Structure of IC [52]**

The typical construction of the system is shown in Figure 2-1 [52]. The main components of this structure are: the die, heat spreader, the package, thermal interface material (TIM), heat sink and the surroundings (ambient). Heat is produced in the die and flows towards the ambient (as the temperature of the die is much higher as compared to the ambient temperature). Thus, the heat flow follows the following path: die -> heat spreader -> package -> TIM -> heat sink -> ambient. The flow of heat is influenced by several factors. The main factors which influence the heat flow are:

1. Thermal resistance (or thermal conductance) of each of the materials in the heat flow path

2. Thermal capacitance of each of the components in the heat flow path

Both of these factors depend on the thermal properties of the material used as well as the physical properties of the materials such as thickness, cross sectional area, etc.

## 2.2 Thermal - electrical analogy

As discussed in [30] there exists a well defined analogy between the heat transfer phenomenon and electrical phenomenon. We can conclude that:

- The thermal resistance (or thermal conductance) can be modeled as an electrical resistance.
- The thermal capacitance can be modeled as an electrical capacitance
- The heat flow is modeled as electrical current
- The temperature at a given point is the electrical voltage in the R-C circuit.

Thus, if we translate the system into electrical equivalent, then we have to use electrical resistances and capacitances. The model using electrical components will look as shown in Figure 2-2.

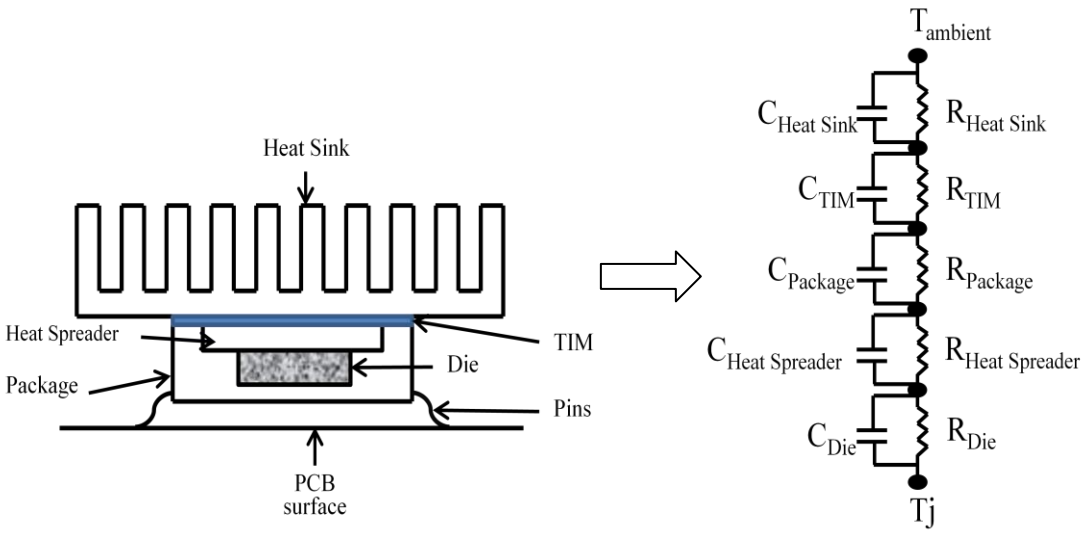



**Figure 2-2: R-C model of the IC [52]**

Using this analogy, solving the heat transfer problem becomes much easier and requires less computation. In case we need to divide the die or other components into smaller blocks, to get the temperature of internal points, the following model shown in Figure 2-3 can be used:

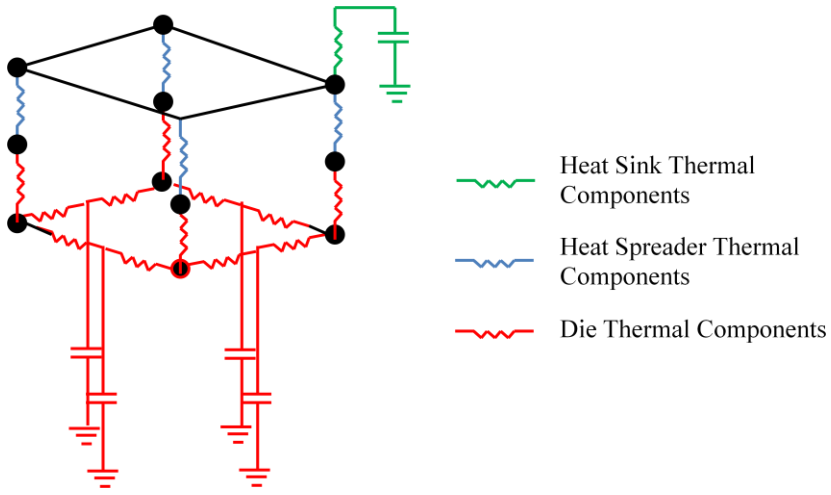



**Figure 2-3: R-C model of the IC and passive cooling solution [52]**

For our environment we need a thermal simulator which allows us to divide the die into multiple components. This is because the IC is divided into several components, the location of which is specified in the chip floorplan. Each of these components dissipates different amount of power. Also, the thermal simulator should be fast and easy to run. The most important point is that it should be possible for us to modify the simulator so as to include different features required for the detailed thermal analysis of future generation ICs. One such thermal simulator which satisfies our requirements is HotSpot [52], developed by university of Virginia.

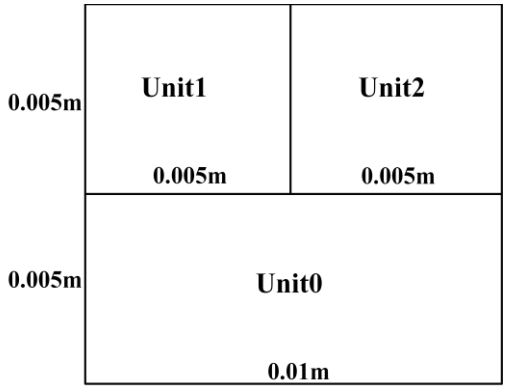
## 2.3 Introduction to HotSpot

HotSpot is a RC thermal simulator i.e. it translates the physical structure of the IC and other components into a RC matrix and then uses the analogy between thermal and electrical phenomena to solve the heat flow problem [52]. We have used HotSpot-5.0

version [53] in our environment (as it is the latest version of HotSpot available). Hotspot models steady state as well as the transient behavior of ICs. In the steady state simulation, only the thermal resistance is critical. So, only the electrical resistance matrix is populated. In case of transient simulations, both R and C matrices are populated and then the differential equations are solved.

Typically HotSpot takes the following inputs:

1. HotSpot configuration file - This file has all the basic parameters needed by HotSpot to construct the R-C matrices for all the components and simulate the RC circuit. It specifies physical parameters such as dimensions of heat sink, TIM, heat spreader. It also specifies the thermal resistances and thermal capacitances of all the components. Other parameters such as ambient temperature, the simulation time step, processor frequency, and initial temperatures of the units are specified in this file.
2. Floorplan file - This file specifies the location of each of the units in the IC. It contains the dimensions of the each of the units in the IC and their bottom-y and left-x coordinates. For example, if the IC die is divided into 3 units as shown below, then the floorplan file will look as shown in Figure 2-4. The extension used for the floorplan file is ".flp".




| #Unit name | width | height | left-x | bottom-y |
|---|---|---|---|---|
| Unit0 | 0.01 | 0.005 | 0.000 | 0.000 |
| Unit1 | 0.005 | 0.005 | 0.000 | 0.005 |
| Unit2 | 0.005 | 0.005 | 0.005 | 0.005 |

**Figure 2-4: Sample floorplan and floorplan input file**

3. Power trace file - The power trace file specifies the power of each of the units over time. The number and names of units specified in the power trace file must match with those in

the floorplan file. The extension used to specify the power trace file is ".ptrace". A sample power trace file for the floorplan shown in Figure 2-4 is shown in Figure 2-5.

| Unit0 | Unit1 | Unit2 |
|---|---|---|
| 0.003 | 1.005 | 0.756 |
| 0.198 | 0.990 | 1.001 |
| 1.130 | 1.100 | 0.540 |

**Figure 2-5: Sample power trace input file**

```
#File Format:

#<Layer Number>
#<Lateral heat flow Y/N?>
#<Power Dissipation Y/N?>
#<Specific heat capacity in J/(m^3K)>
#<Resistivity in (m-K)/W>
#<Thickness in m>
#<floorplan file>

# Layer 0:Silicon
0
Y
Y
1.75e6
0.01
0.00015
new.flp

# Layer 1:thermal interface material (TIM)
1
Y
N
4e6
0.25
2e-5
new.flp
```

**Figure 2-6: Sample layer configuration input file**

4. Layer configuration file - The layer configuration file is used to specify the characteristics of different layers in the IC. For example, if the chip is a 3D IC, then multiple silicon layers are present. These silicon layers can have transistors and can dissipate power. If the IC under consideration is not a 3D IC, then just 1 layer will

dissipate power. All this information is specified in the layer configuration file. The extension of this file is ".lcf". A sample layer configuration file is shown in Figure 2-6.

HotSpot populates the required R-C matrices and then outputs the following files in which the values of temperature are stored.

1. Steady state temperature output file – HotSpot can produce two output files. The first is the steady state temperature file. This is generally denoted by the ".steady" extension. This file contains the results of DC simulation carried out by HotSpot. In case of DC simulation, the thermal capacitances are ignored and only the thermal resistances are taken into account.
2. Transient temperature output file - If we desire to have the values of temperature after each time interval, then we can dump the temperature values into a file called the transient temperature file. This file is denoted by the ".ttrace" extension. In our setup, this file is the most important file as it contains the value of temperature of each of the units in the floorplan and at every time interval. We can analyze this file to take decisions regarding the thermal profile of the processor.

## 2.4 Flow of HotSpot

As discussed earlier, HotSpot is a RC thermal simulator. Hence, it needs to first build the RC network. It takes the floorplan file and physical parameter from the "hotspot.config" file and creates the required RC matrices. HotSpot then reads lines from the power trace file and calculates the rise in voltage (i.e. temperature) of each unit in the floorplan using a differential equation solver. It keeps on dumping transient temperature values after each time step. When all the lines in the power trace file are exhausted, it dumps the steady state temperatures into a file and exits. The flowchart of HotSpot is shown in Figure 2-7. The pseudo-code is shown in Figure 8-1 (in Appendix).

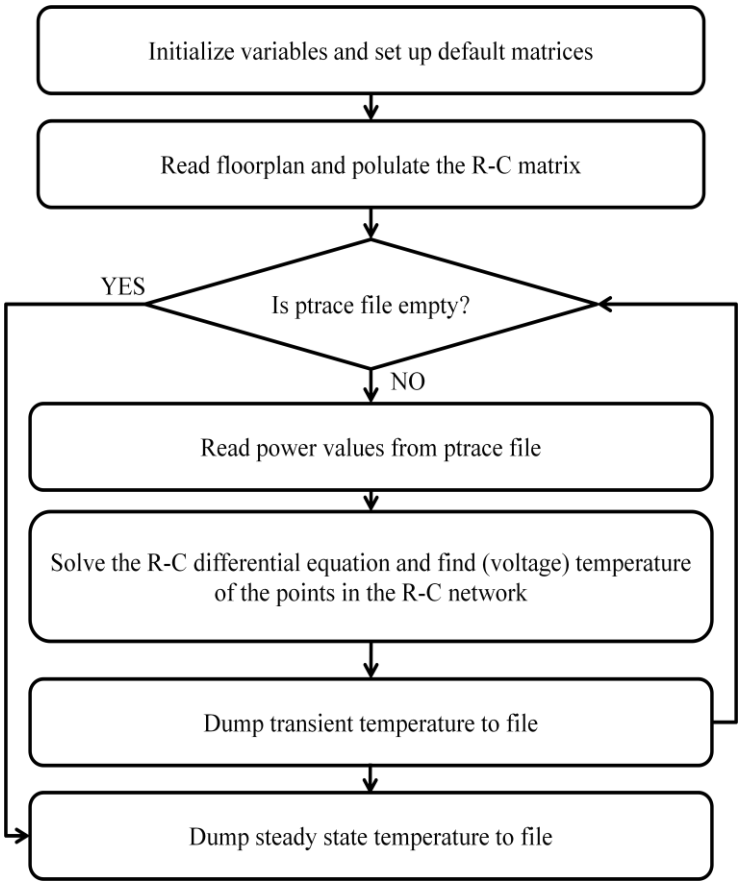



**Figure 2-7: Flow diagram of HotSpot**

## 2.5 Additional features added to HotSpot to model important parameters

### 2.5.1 Leakage power – temperature interaction

The sub-threshold leakage power in CMOS technology is dependent on the technology node and operating temperature. For different technologies, the dependence of sub-threshold leakage power on temperature is different. The dependence characterization can be done using simple Hspice simulations for the target technology. Typically the leakage power of a CMOS transistor is exponentially dependent on the operating temperature. After running Hspice simulations on 45nm Predictive Technology [54], we characterized the leakage power with respect to temperature. We observe that a curve fitting done using a quadratic equation gives sufficient accuracy for modeling the leakage temperature interaction.

In order to introduce this feedback loop into HotSpot, certain changes were made. New local variables ("leak_coeff" arrays) which hold the coefficients of the quadratic equations used to model the interaction were introduced. The actual coefficients of the quadratic equation were stored in a file. These are then read into the "leak_coeff" arrays and stored locally in order to increase speed of execution. The leakage power is computed using the instantaneous values of temperature and also the corresponding leakage power coefficients. The leakage power is then added to the dynamic power which is obtained from the power trace file. This total power is then used to calculate the temperature during the next time interval.

It should be noted that the power trace file contains power of each of the units specified in the floorplan file. Similarly, leakage power is calculated for each of the units. This implies that each unit should be associated with the corresponding "leakage coefficients". Thus, leakage coefficients are specified for each of the units in the design. To keep the leakage updates independent of the actual sampling interval or migration interval which HotSpot takes in as an input, we calculate the number of steps required to update the leakage power values. This is done using a function which calculates the number of update intervals as follows: If the sampling interval of hotspot is 3.3ms, and the leakage update interval is 3.3 us, then the number of steps required is 1000. Similarly if the sampling interval changes to 33.3 us, then the number of steps for leakage update is modified to 10. Thus, we can keep the leakage update interval constant over all simulations.

Typically, leakage power-temperature curves of cache blocks and core blocks are different. Hence, we need to have different sets of equations for core blocks and cache blocks to represent the leakage power-temperature interaction. This is done by keeping the equations for each type of block in different files. The file used for storing the core leakage equations is called "core_leakage_coeff_file" and the file used to store the cache leakage equations is called the "cache_leakage_coeff_file" (if nothing is present in any of these files, the coefficients are assumed to be 0). These coefficients are then stored into local arrays called "leak_coeff" arrays as specified in the previous section. The flow diagram and pseudo-code for implementing this feature is shown in Figure 8-2 and **Error! Reference source not found.** (in Appendix) respectively.

### 2.5.2 Inter-die and intra-die process variation

As discussed earlier, another important component of thermal modeling is emulating the process variation. This process variation is of two types viz. inter-die variation and intra-die variation. Additional modifications are required to the HotSpot code in order to implement this feature. A way to model such inter-die process variation is to have a separate equations to represent the leakage power – temperature interactions for different dies. But within the die, all the blocks will have the same interaction curve. In our setup, we have a file (leakage_coefficient_file) which stores the coefficients of the quadratic equations which represent the leakage power values at different temperature values. As discussed earlier, we have different files to store leakage coefficients for cache blocks and core blocks. The different coefficients are randomly tied to each of the units specified in the floorplan. If we have just one leakage equation in the cache_leakage_coeff_file and one equation in the core_leakage_coeff_file, then this same equation will be assigned to all the cache and core blocks respectively. This means that all the core blocks will have same leakage power profile (if temperature is same), and all cache blocks will also have the same leakage power for the same temperature value.

Thus, in order to simulate inter die variation, we can put just one equation each in the “core_leakage_coeff_file” and the “cache_leakage_coeff_file” files. For the entire simulation, the same value will be used for all blocks. In the next simulation, we can change the value of the leakage equation and run the simulation. Thus, each simulation will emulate a die with different leakage profiles. In this way, inter-die variation can be simulated in our environment.

If we put multiple equations in these files, each of the blocks in the floorplan will be randomly assigned a leakage equation from these files. Generally, the intra-die process variation has a certain degree of correlation. But for simplicity, we have not included this behavior in our setup. Thus, we can simulate random intra-die process variation using the same set of files and arrays as used for intra-die process variation. One limitation of the current setup is that we cannot simulate inter-die and inter-die variation simultaneously. The general flow and pseudo-code which is used to implement both intra-die and inter-

die process variation in the setup is shown in **Error! Reference source not found.**8-5 & Figure 8-4 (in Appendix).

### 2.5.3 Thermal sensors

**Number and location of thermal sensors:**

An important aspect of getting information about the temperature of different locations in the chip is the number of temperature sensors, their location and the sensor accuracy. It is not practical to have a very large number of sensors in the chip as they consume power and real-estate. As a result, the number of sensors placed on the die is typically low. Also the temperature sensors are placed at optimal locations so that temperature can be effectively monitored in the chip. Sometimes, if we design a chip using Intellectual Property (IP) provided by a third party, we may not be able to place temperature sensors inside such IPs (if we are using hard IPs). This imposes a limitation on the location and number of thermal sensors. Also, if the sensors are not accurate, errors are introduced in the temperature readings causing unreliable thermal profile information. These limitations have to be considered while analyzing the temperature profile of the IC.

In our setup, the number of sensors and the locations of these sensors which are placed in the chip can be modified by changing the "sensor_location_map" file. This file stores the mapping of the units which have sensors present. For example, if the floorplan file has 4 units, then the "sensor_location_file" will also have 4 units. But if a sensor is not present in the unit, a "0" will be used to indicate this fact. A "1" indicates that a sensor is present in the unit. This sensor file is then loaded into a local array called "sensor_location_map". When we scan the "temp" array to find out the instantaneous temperature of a unit, we use the "sensor_location_map" array as a mask. Thus, we will be able to get temperature values of only those units which have sensors present.

The sensors are assumed to be in the center of the unit. If we need that the sensor be located at a position other than the center of the unit, additional modifications are required. HotSpot divides each unit into a grid. We can extract the temperature of the exact point in the grid using an internal array. In this way we can place the sensor

anywhere in the chip. The “sensor_location_file” has to be prepared before the start of the simulation.

**Sensor sampling interval:**

The temperature sensors which are placed on the die have a certain bandwidth i.e. they can output the value of the sampled temperature after a predefined interval of time. Generally, more the sampling interval, more accurate is the sensor reading and vice versa. The bandwidth of the sensors can play an important role in case we need fast sampling of the temperature values in the chip. The sampling interval of the sensor can be specified in our current setup using the local variable called “sensor_sampling_interval”. We will be able to read temperature values only after the sampling interval time. Otherwise, HotSpot will keep on running and we will not be able to get any information about temperature of the chip. The sensor sampling interval has important implications on the policy used for thermal management. We will discuss these implications in Chapter 3.

**Sensor accuracy:**

In order to increase bandwidth of the temperature sensor, some sensors compromise on the accuracy of the sampled temperature. This error in the sensor reading can also be included in the simulations. This accuracy may be critical if we want to have a perfect picture of the thermal map of the IC. The accuracy is specified either in terms of percentage of the actual value or in terms of actual maximum error possible. If the actual maximum error is specified, then we use a random number generator to generate a value between –ERROR and +ERROR and this value is then added to the actual temperature reading. For percentage error, a random number is generated between –percent_error and +percent_error and the corresponding value is then added to the actual value of temperature (which is obtained by sampling the array which stores the instantaneous temperature values internal to HotSpot).

### 2.5.4 Adaptation of number of active cores

In case of manycore processors having large number of cores, it may not be possible to keep all the cores active due to thermal limitations. As a result, we need to adapt the number of active cores depending on the temperature of the chip. This online adaptation of the number of active cores is especially important when we have ICs with high process variation. In case of low $V_t$ corner ICs, the leakage power will be much higher and the resulting temperature will also be much higher. Hence, the number of active cores needs to be reduced. In case of high $V_t$ corner, we may be able to keep more number of cores active. This adaptation has been added to the existing HotSpot framework. The flow diagram of the implementation is shown in Figure 8-7 (in Appendix).

### 2.5.5 Dead cycles

When we switch off a certain core and switch on another core to complete the remaining task, we assume that it happens in "zero" time. In practice this is not the case. Switching on the core requires a finite amount of time. This is because the internal node capacitances and gate capacitances need to be charged before the core can resume computation. During this charging up time, the core cannot perform any computation and has to wait until all the internal circuit nodes have stabilized. Moreover, if the cores which are involved in such a scenario do not share a cache, then we may need to transfer the state of the cache as well to avoid cold-start misses. This cache state transfer operation requires a certain amount of time to complete. During this time, no computation can be done by the newly activated core. In our setup, we call this interval as the "dead cycles". This interval has to be considered while modeling the thermal profile.

The power consumed during this interval is the switching power of the new core. The "power" array which stores the instantaneous values of the power needs to be changed in this case. The pseudo-code for this feature is shown in Figure 8-6**Error! Reference source not found.** (in Appendix).

### 2.5.6 Multiple floorplan structures and orientations

As mentioned earlier, semiconductor ICs are typically divided into multiple units and these units are placed in a specific way on the die. This is called the floorplan of the chip. We consider homogeneous manycore processors in the current environment. It consists of cores in X and Y directions. As the processor is homogeneous, all the cores are identical. Initially, we assume that the cores are not further divided into smaller units. The entire chip with such non-divided cores is shown in Figure 2-8.

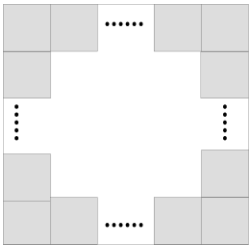


Figure 2-8:Floorplan of a homogeneous manycore processor

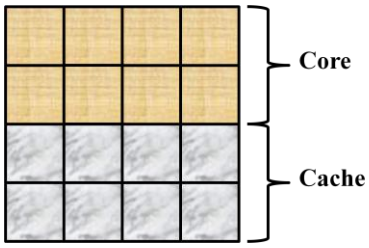



Figure 2-9: Floorplan with cache and core blocks (orientation1)

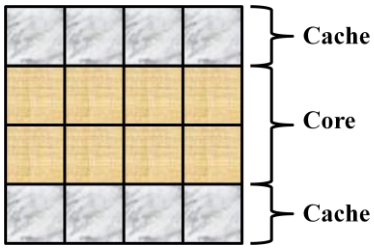



Figure 2-10: Floorplan with cache and core blocks (orientation2)

In case we need to divide each core into smaller units, we assume that the core has two types of units viz. core units and cache units. Cache occupies half of the total area of each core. This assumption is based on observations made on recent microprocessor chips (in recent microprocessors, half of the total area is accoupied by last level of on chip cache). The remaining half of the chip is dedicated to the core units. The cache can be organized into different structures as shown in Figure 2-9 and Figure 2-10. The different organizations enable us to model different cache organizations such as shared cache or private cache.

### 2.5.7 Modeling hotspots in the core

Typically, some of the units in the core will dissipate much more power as compared to the other units (for e.g. ALU and register file will dissipate much more power than the branch predictors). If this is the case, then we need to model such hotspots which may be present in the individual cores.

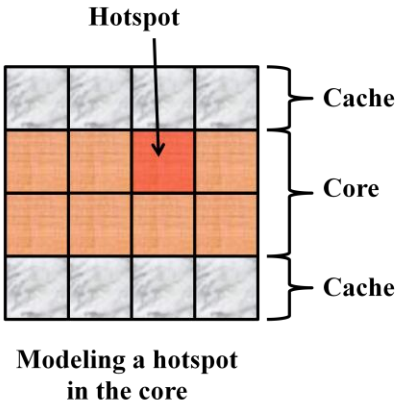



Figure 2-11: A single hotspot present in the core

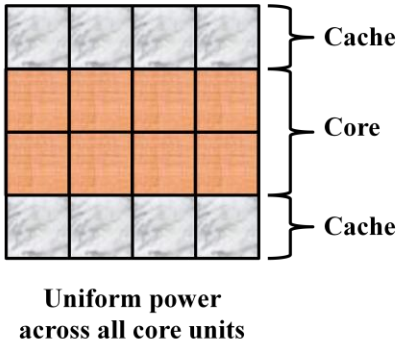



Figure 2-12: No hotspot present in the core due to uniform power density

If we know that all the units in the core dissipate approximately the same power, then we need not have hotspots in the chip. This scenario can also be simulated in our enviroment. In this case, we need to specify that the units dissipate almost equal amount of power leading to uniform power density within the core. If we want to model a hotspot in the core, we need to specify the location of the hotspot and also the power density of the hotspot to be modeled. All these parameters are specified in the "flpconfig" file. This information is then read by HotSpot and the dynamic power is assigned accordingly.

### 2.5.8 Thread mapping

In real multiprocessor systems, different cores run different threads. After a thread on a core finishes, a new thread is assigned to the core. We try to model this behavior by binding threads to the cores in the manycore chip. To do this, we need the power profile of the thread. This power profile file is then tied with the core. To get power profiles of different threads, we run the SPEC2006 benchmark suite on an architecture simulator called Zesto [55]. We assume that different cores in the manycore chip will run threads which are similar to the programs in the benchmark suite. So, we tie the power trace files obtained from Zesto with each core in the IC. To do this, we construct an array for each core. This array holds the value of power for each time interval. We have used local arrays in order to decrease execution time (as time to access the arrays will be less as compared to reading values from a file. But memory requirement will increase

significantly as we need to store the local arrays in the memory). We can also control the number of cores which are active by assigning a "zero" dynamic power to cores which are inactive.

## 2.6 Extending the current framework

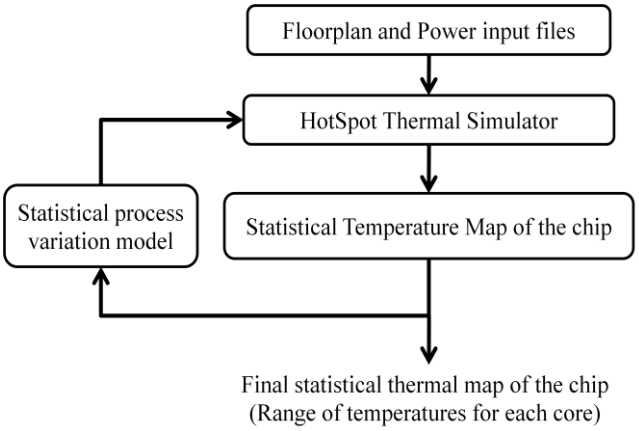



Figure 2-13: Flowchart to implement statistical process variation models

While including the temperature – leakage power interaction, our implementation requires the user to specify the coefficients of the equations representing this interaction. So, if we need to model the process variation in the current environment, we have to specify multiple equations and also have to make multiple runs. In order to make the analysis of process variation faster, we can include statistical models of the interaction curves. Such statistical analysis will give us a statistical temperature range which can be used to analyze the effect of process variation. The flowchart to explain this implementation is shown in Figure 2-13.

In the current implementation, we have considered only homogeneous (symmetric) manycore processors. In practice we may use processors which have different types of cores integrated on the same chip. Such heterogeneous (asymmetric) processors also need to be included in this modeling framework. An immediate extension of the current implementation can be to model thermal behavior of asymmetric manycore processors and include all the current features like thread swapping, etc for such

processors. Similarly, the current framework cannot model 3D ICs. 3D ICs are being explored extensively in order to design future generation processors. We need to include features which can be used to model and manage the thermal fields in such 3D ICs.

In order to model the entire system, we need to integrate architecture simulators which can directly feed in power profiles of different applications used in practice. In addition to the generation of power trace files, we need to add components which can analyze the performance of the entire system under thermal constraints and under different thermal management techniques. As thermal behavior of the manycore processors is closely related to the reliability of the system, we need to integrate models which can predict the reliability of the system.

## 2.7 Summary

In this chapter we have discussed the framework we have developed for thermal modeling of manycore processors. We have also discussed all the additional features which have been included in the framework in order to make it more flexible in terms of modeling different scenarios. We will now use the developed framework to demonstrate how the different features can be used to model different thermal scenarios.

# 3 THERMAL MODELING AND MANAGEMENT

In this chapter we discuss how our modeling environment can be used to analyze the thermal behavior of many core chips.

## 3.1 Metrics used for evaluation migration policies

**Maximum chip temperature:**

The most important metric that determines thermal reliability is the maximum temperature in the chip. If the temperature in any part of the chip exceeds the thermal design point temperature, the devices in that part of the chip may suffer permanent damage. Our primary goal is the control this maximum temperature so that the reliability of the chip is maintained at all times. Maintaining the maximum temperature in the chip under a specific threshold may be difficult in some cases when we have very less number of sensors in the processor or the sensors are highly inaccurate. Other conditions under which the temperature may rise beyond the desired value is in case of chips in the very low $V_t$ corner and if the sensors have a very low sampling rate.

**Maximum spatial gradient and spatial difference:**

In case of chips with multiple heat generating locations (typically manycore chips) the temperature at different locations in the chip will be very different. As a result, there will be a gradient of temperature between the cores of the chip. A lower spatial gradient helps improve package reliability and cooling efficiency. Also, with a lower spatial gradient, the cores will “age” at the same rate thus leading the approximately the same life time of the cores. This also increases the system reliability. The spatial gradient is calculated as the difference in temperature between adjacent cores (in both the X and Y directions). The maximum value of this difference is the maximum spatial gradient. Thus, spatial difference is the difference between the maximum temperature in the chip and the minimum temperature in the chip.

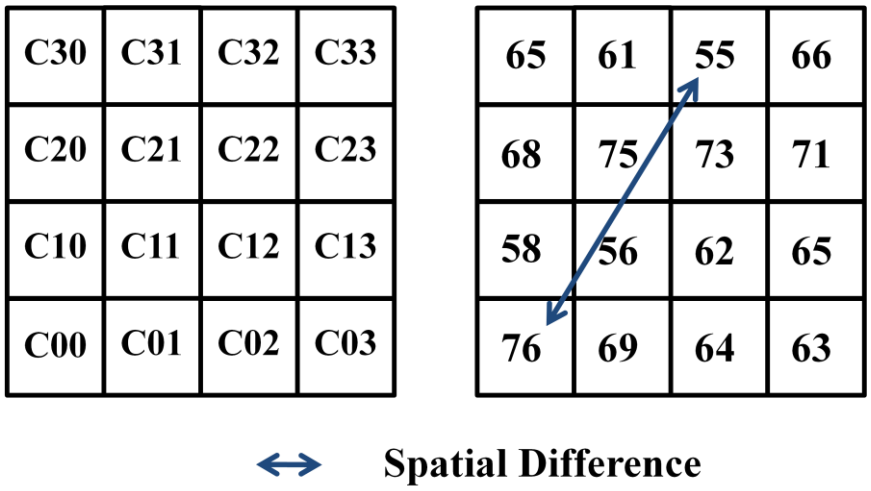



Figure 3-1 Spatial Difference and Spatial Gradient

Figure 3-1 shows the floorplan for a 16 core chip. It also shows the temperatures of each of the cores at time 't'. The spatial gradient can be found out using the following formula:

$$(Spatial\ Gradient)_x = max\left\{\frac{|C_{i+1,j} - C_{i,j}|}{distance\ between\ cores}\right\}, i = 0\ to\ 2, j = 0\ to\ 3$$

$$(Spatial\ Gradient)_y = max\left\{\frac{|C_{i,j+1} - C_{i,j}|}{distance\ between\ cores}\right\}, i = 0\ to\ 3, j = 0\ to\ 2$$

$$\text{Maximum Spatial Gradient} = \max((\text{Spatial Gradient})_x, (\text{Spatial Gradient})_y)$$

$$Spatial\ Difference = Maximum\ chip\ temperature - Minimum\ chip\ temperature$$

**Maximum temporal gradient and temporal difference:**

As the power dissipation at a location in the chip varies over time, so does the temperature. Therefore, run-time power variation in the chip results in thermal cycles. The on-chip power management techniques used to control the thermal field in the chip can also cause thermal cycles. The reliability of the devices, interconnects, insulation, package, etc are dependent on the thermal cycles. The higher the amplitude and frequency of these thermal cycles, the more detrimental it is to the system reliability as a whole. So, we need to keep a control over the thermal cycles of the cores. The metrics which are used to measure the severity of the thermal cycles are the temporal gradient and temporal difference. Temporal gradient of a core is the difference in temperature of a

specific core in successive small time intervals. Thus, temporal gradient needs to be computed on a per-core basis. The maximum value of temporal gradient amongst all the cores is the chip temporal gradient.

The temporal difference is the maximum difference between the maximum and minimum temperature of a core over a long time interval. The importance of the temporal difference arises from the fact that it gives an idea of the low frequency components of the thermal cycles. In some cases (such as package reliability), temporal difference may be of more importance as the high frequency components of thermal cycles may not have a significant impact in these specific cases. The temporal metrics can be illustrated as shown in Figure 3-2. The temporal gradient and temporal difference are calculated after initial simulation period (i.e. after the initial temperature rise period).

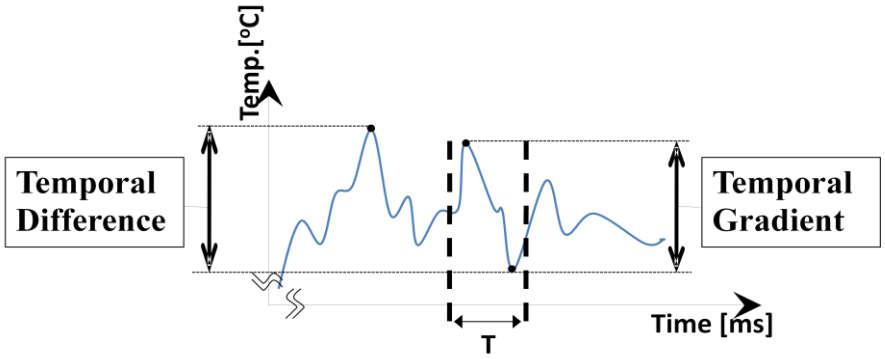



Figure 3-2 Temporal gradient and temporal difference

**Number of migrations:**

As discussed earlier, migrating cores after specific time intervals can serve as one method to manage the temperature in the chip. Each time migration happens, the performance takes a hit. Also additional power may be required to transfer the state of the core to the new active core in addition to the power required to charge up the internal node capacitances in the new active core. Thus, the number of migrations may serve as an important metric to analyze the performance degradation associated with the management policy. Each time a core is switched from active to inactive state, the number of migrations for that core increases by 1. Thus the number of migrations is stored on a per-core basis. At the end of the simulation, we can add up all the migration counters to get the total number of migrations during the simulation period.

## 3.2 Application of the model for thermal management

**Thermal management using heat redistribution and thread migration:**

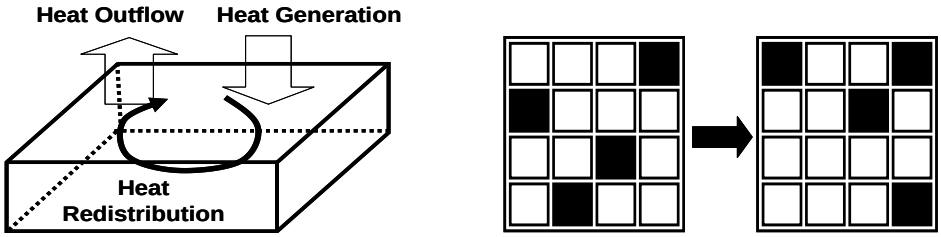



Figure 3-3: Heat redistribution illustration.

For a constant heat generation and outflow, temperature can be controlled by redistributing the generated heat in the chip over space and time. This redistribution is enabled by turning off some of the cores periodically and turning on other cores at other cooler locations (i.e. by moving the computation). In this section, we discuss the application of the developed framework in managing thermal field using heat redistribution. In our modeling framework, heat redistribution essentially means that we need to change the power matrix used by HotSpot. This power matrix is stored internally to hotspot. We need to change the values of the power associated with each core in order to enable thread migration. By using the power multiplexing feature of our environment, we can observe that the temperature of the many core processors can be reduced significantly.

The migration policies can be of two main types, viz. reactive policies and proactive policies. Reactive policies enable the control mechanism only when the metric they are monitoring crosses a threshold. For example, most of the current processors monitor the temperature of the chip. When the temperature crosses a threshold value, the frequency and voltage of the processor is reduced so as to reduce the temperature of the chip. On the other hand, proactive techniques enable core migration even if the metric they are monitoring has not crossed a threshold. The migration happens after a pre-define time interval called “migration interval”. The effect of this is that heat redistribution is enabled and the temperature of the chip takes a longer time to reach the threshold value.

In the worst case, if the temperature reaches the threshold value, we have to take severe corrective action in order to save the chip from damage due to high temperatures.

### Implementing reactive and proactive policies:

In order to implement any reactive policy, we need to change the internal "power" array used by hotspot. This change in the "power" array has to happen only if the metric being monitored crosses the threshold. In our case, all metrics are related to temperature and hence we need to have temperature data before enabling reactive migration. This temperature data is only available after the "sensor_sampling_interval". Thus, while implementing reactive policies, we internally change the values stored in the "power" array to simulate either DVFS of on-off type control after each "sensor_sampling_interval". One important modification to be done while implementing reactive policies is that now the dynamic power values are not read from a power trace file (power trace file is used only for initialization). Instead, the power values are generated internally. We also introduce the max_runtime variable while implementing this feature. This is used to exit the main while loop. In order to make this implementation possible, the original flow of hotspot needs to be modified. The flow diagram used to implement the reactive policies is shown in Figure 3-4 and the pseudo code in shown in Figure 8-8 (in Appendix).

The implementation of proactive policies is similar to reactive policies. The main difference is that the migration happens after every "migration_interval" (instead of sensor_sampling_interval in case of reactive policies). The flow diagram shown in Figure 3-4 is used to implement the proactive migration policy. The only difference between implementing reactive and proactive policies is that, in proactive policies we need not stall the migration until the thermal sensor samples are available. Instead we migrate after a predefined time interval called "proactive_migration_interval". The pseudo-core for implementing this policy is shown in Figure 8-9 (in Appendix).

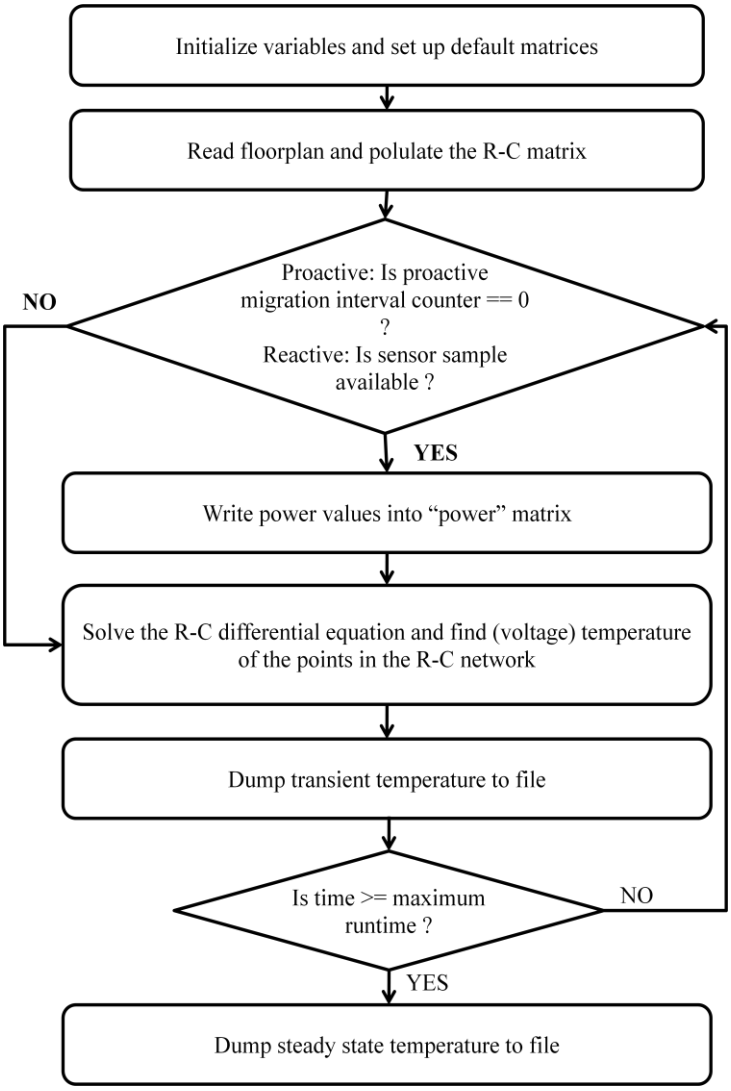



**Figure 3-4: Flow diagram for implementing the proactive and reactive migration policies**

In the following sub-section, we will show how the modeling environment can be applied to model the critical effect of proactive and reactive migration.

### 3.2.1 Modeling the effect sensor properties on reactive management policies

Reactive power multiplexing depends on the fact that we have information of all the cores in the chip. This information is then used to decide if any of the cores violates the thermal constraint. In case we do not have information about the temperature at each and every core in the chip, we may not be able to manage the temperature in the chip

correctly. If we have a very less number of sensors in the chip, we may not have information about some of the cores leading to high temperatures in those cores. This phenomenon is shown in Figure 3-5. Here we see that reactive power multiplexing may not be effective if we do not have adequate sensors in the chip (or the sensors are not places at optimal locations throughout the chip) as we do not get correct information about the chip temperature. In the following simulation, the migration threshold is 70$^{o}$C. As expected, if we have very less sensors in the chip, reactive migration does not work as desired due to lack of proper temperature information.

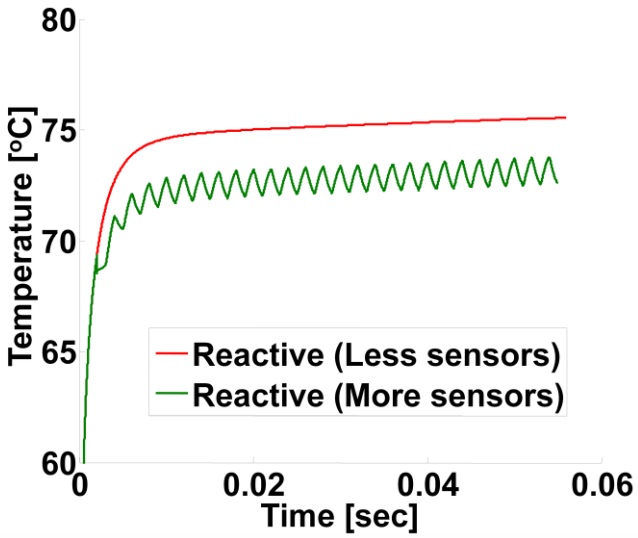



**Figure 3-5: Effect of number of on-die thermal sensors**

In addition to the number of sensors present in the chip, the sampling interval of the sensors has significant effect on the effectiveness of the migration policy implemented. The sensors which are present on the die need a specific time to sample the temperature values. This sampling interval is dependent on the design of the thermal sensor. Also, the thermal sensors have a certain associated with the sampled value of the temperature. Some sensors compromise the sampling interval in order to get a very high accuracy, while others have a very small sampling interval but live with a wider error range. The sampling interval of the sensors as well as the error associated with the sensor reading has significant implications on the choice of the thermal management policy.

In case the thermal sensors have a very large sampling interval (as compared to the time required for the temperature to rise significantly. This will in turn depend on the power dissipated in the unit), it makes more sense to use proactive migration policies. This will make the migration independent of the sampling interval of the sensor as well as the error associated with the sampled temperature.

### 3.2.2 Modeling proactive policies

The current framework can also be used to model proactive migration policies. In proactive migration, the migration happens after specific time intervals called "proactive migration intervals".

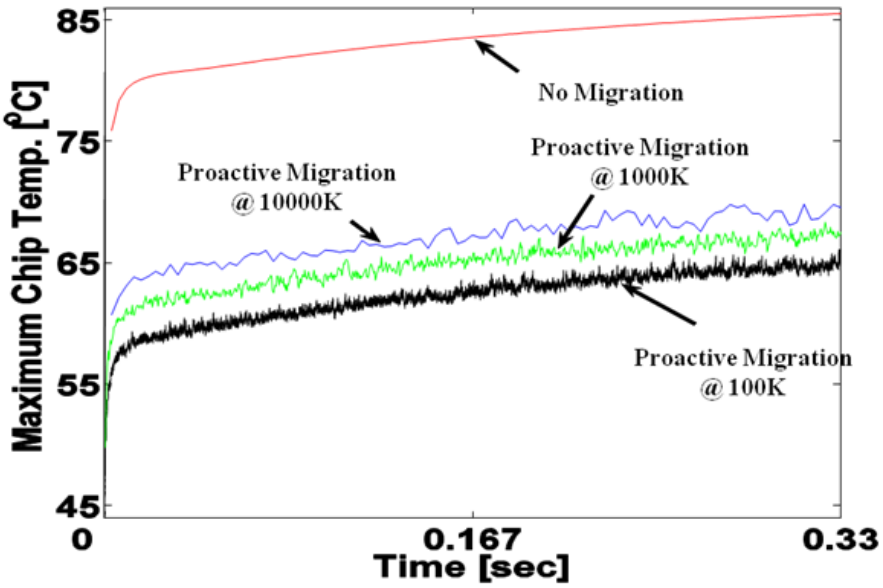



Figure 3-6: Effect of different thermal management policies on thermal profile

In case we use proactive migration policies as our primary management policy for temperature control, the time after which we redistribute the power in the chip will affect the temperature profile of the chip. If the temperature-rise per unit time of the individual cores is high then the migration interval needs to be small in order to keep the temperature under control. But in this case the migration cost can be high. But if the power dissipation in the cores is low, then the temperature rise will also be small. In such

cases, we can live with a large migration interval. We have included this feature in our modeling infrastructure.

In our environment, we often measure the migration interval in terms of number of cycles. For example, if we have the operating frequency of the processor as 3GHz and the migration interval is 3.333 µsec, then the migration interval is 10K cycles. So a migration interval of 500K means that we enable migration after every 500K cycles at 3GHz i.e. 0.166 msec. During this work, we assume that the processor frequency is 3GHz unless we specify otherwise.

A critical effect of the migration interval is the maximum on-chip temperature achieved. Figure 3-6 shows the variation of maximum on-chip temperature with respect to the migration interval used. The sampling interval also affects the other thermal metrics. As we keep on decreasing the migration interval, all the cores get a smaller time interval to heat up or cool down. As a result of this, the thermal profile of the chip becomes more uniform as compared to a larger migration interval. It is obvious that a more uniform thermal profile will have a better spatial difference and temporal difference. This is shown in Figure 3-7. As expected, the higher the sampling interval, worse is the value of the spatial difference and temporal difference.

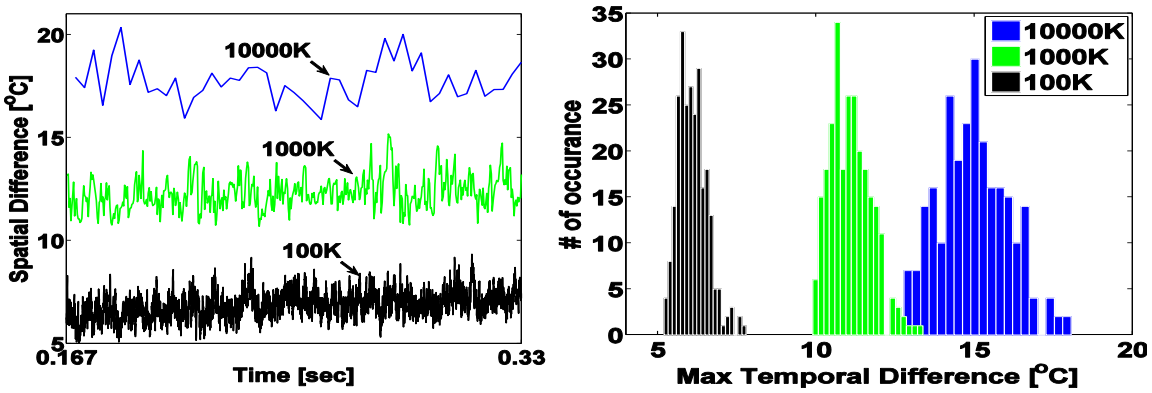



Figure 3-7: Effect of migration interval on Spatial and Temporal Difference

### 3.2.3 Modeling the effect of number of active cores

Future generation manycore chips are expected to have a large number of cores. But it is likely that only some of the core will be active at a given time. If a small percent

of the total cores is active at a given time, the total power density of the chip decreases. This causes temperature to rise at a much slower rate. In such cases, migration interval can be higher. If we consider that the migration interval is constant then it is obvious that more the number of active cores, more will be the temperature rise due to the higher power density. This behavior of the temperature profile with respect to the number of active cores can also be analyzed using our modeling environment. Figure 3-8 shows the effect of varying the number of active cores on the temperature of the chip. The migration interval in this case was kept constant at 500K cycles. The current environment can also be used to evaluate the thermal behavior of the manycore chip if the number of active cores varies over time. This is illustrated in Figure 3-8.

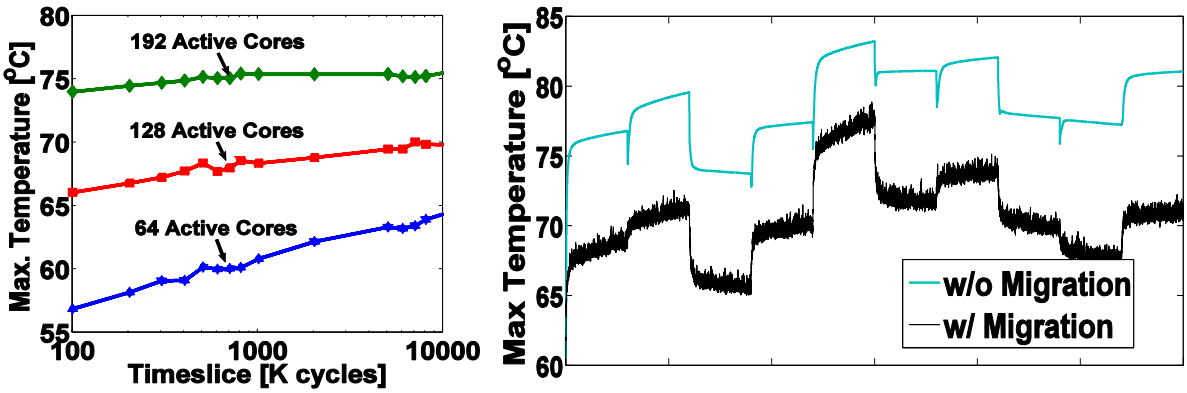



Figure 3-8: Effect of number of active cores on maximum chip temperature

### 3.2.4 Modeling process variation

Process variation can also be simulated in our environment. The effect of process variation is shown in the following figure. This simulation was carried out with the same proactive migration interval. As expected, for the low Vt process corner the value of maximum chip temperature is higher. For the low Vt corner, the leakage power is much higher. Also the dependence of leakage power on the operating temperature is much stronger. As a result of this increased leakage power and stronger feedback, the power density on the die is much more leading to higher values of chip temperature.

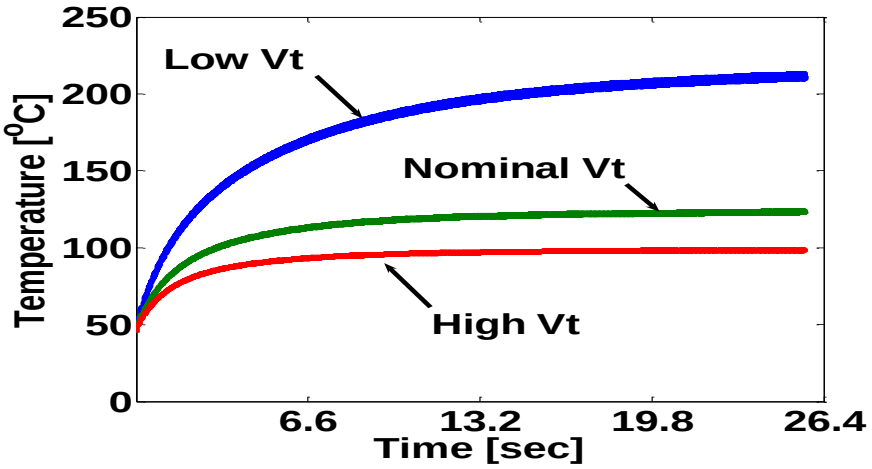



Figure 3-9: Effect of process variation on maximum chip temperature

### 3.2.5 Modeling different proactive management policies

We now use the simulation framework to model different migration policies

**Random migration:**

As the name suggests, random migration selects a random set of cores each time we need to migrate. Random migration does not pay attention to the thermal metrics such as thermal proximity, thermal compactness or migration distance. As a result, we can see scenarios when the thermal characteristics of the chip are worse than the average case. Random migration can be effective in proactive migration techniques where information from the temperature sensors may be neglected for deciding the next set of active cores.

The number of active cores can be controlled easily using this random migration technique. This technique gives satisfactory results in terms of thermal performance, but the performance degradation caused may be large as we do not use any information about the migration distance or total number of migrations or other performance related metrics. Our infrastructure uses a C-function which generates a set of cores randomly which need to be turned on randomly. This C-function is based on the rand() function of the 'C' language. The distribution of random numbers generated by this function is "uniform". As a result, all the cores will be active for approximately the same amount of time during the entire simulation period.

**Cyclic migration:**

Random migration selects the next set of active cores on a random basis and may give rise to scenarios where the thermal compactness and thermal proximity are high. At the other end of the spectrum, we have policies which consider the thermal proximity and thermal compactness metrics to decide the next set of active cores. One such policy is the cyclic migration policy. In the cyclic migration policy, we divide the entire chip into groups of 4 cores each. These 4 cores must form a square or rectangle between themselves. This is shown in Figure 3-10. Here we consider that we have a 16 core chip. So we divide the chip into 4 groups having 4 cores each.

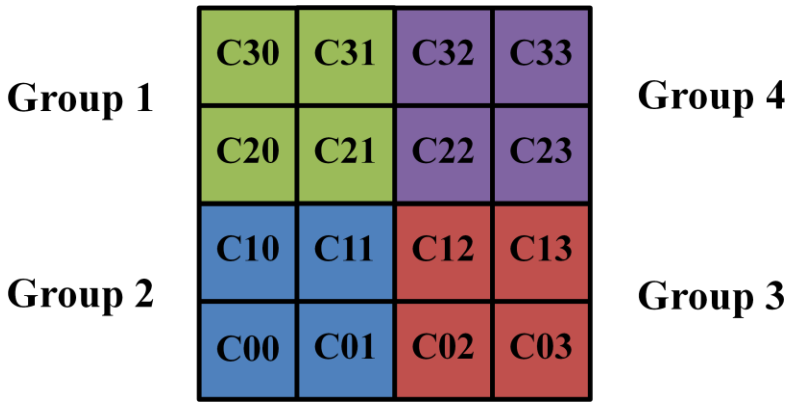



Figure 3-10: Cyclic migration group formation

In the cyclic migration policy the number of active cores is restricted to 4 options viz. 25%, 50%, 75% and 100%. This means that the number of active cores at any given time may be either 25% of the total cores or 50% of the total cores or 75% of the total cores or 100% of the total number of cores in the chip. This is a major drawback of the cyclic migration policy. If we consider that 25% of the total cores are active at any given time, the migration pattern will look like we are shifting the cores in a cyclic fashion. This is shown in Figure 3-11. We observe that we select a checker-board configuration to select the next set of active cores. This configuration has excellent thermal compactness and thermal proximity properties. If we consider that a group of 4 cores shares a cache, then the state migration cost is also reduced as the newly active core has access to state of the cache through the shared cache configuration.

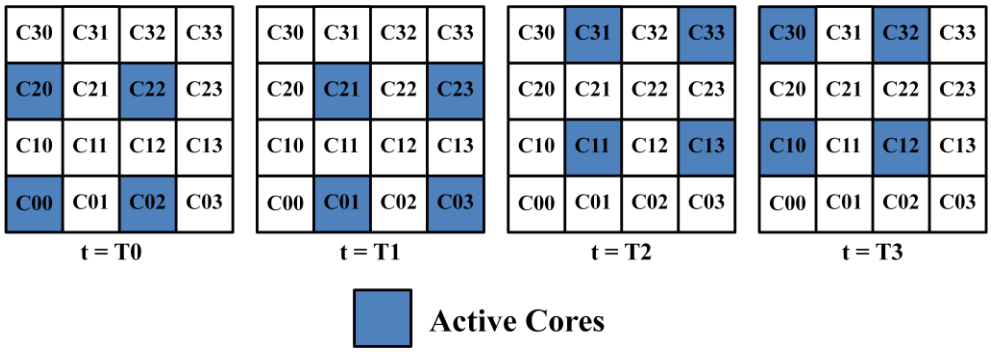



Figure 3-11: Cyclic migration illustration for 25% active cores

**Global coolest replace:**

One of the most widely discussed policies in the literature is the global coolest replace. As the name suggests, we migrate to a core which is coolest at that instant in time. There are many variants of the global coolest replace policy. In a proactive migration, we first sort the cores in terms of their temperatures. Then, we migrate the 'N' hottest cores to the coolest 'N' cores.

It is also important to analyze the other thermal metrics in order to gain a complete understanding of the policies which we have implemented in the modeling environment.

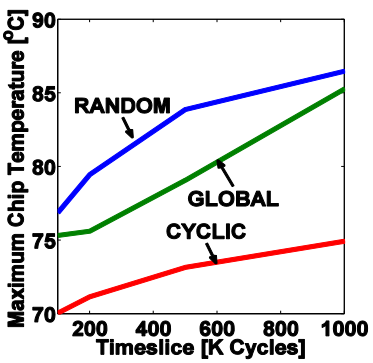



Figure 3-14: Effect of migration policy on maximum chip temperature

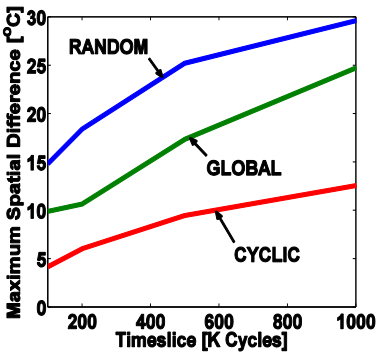



Figure 3-13: Spatial difference for different migration policies

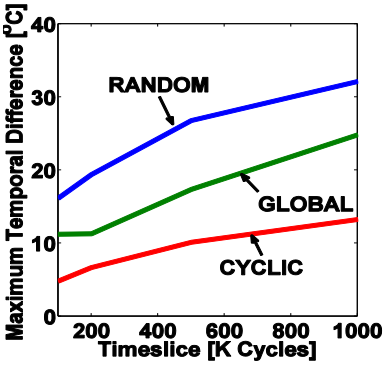



Figure 3-12: Temporal difference for different migration policies

Figure 3-12, Figure 3-13 and Figure 3-14 show the comparison of maximum chip temperature, spatial and temporal difference between all the 3 policies respectively. From the figures, we can conclude that the cyclic migration policy is most effective form the

thermal perspective. The random policy fares worst than the cyclic and global coolest policy as it does not take decisions depending on the thermal field in the chip.

## 3.3 Modeling thermal management during burn-in

Another interesting application of the modeling environment is in the field of burn in testing. Burn in testing is carried out so that the faults in the chip are detected at a very early stage. Burn in testing a typically carried out at elevated temperatures and supply voltages. This causes the failure mechanisms to accelerate causing faults to occur very early during the testing period. As the burn in tests are carried out at elevated temperature and voltages, the power dissipation (and subsequently the power density) is much higher as compared to the normal period of operation. Also the leakage power is higher due to higher temperatures and the positive feedback is stronger in such test conditions.

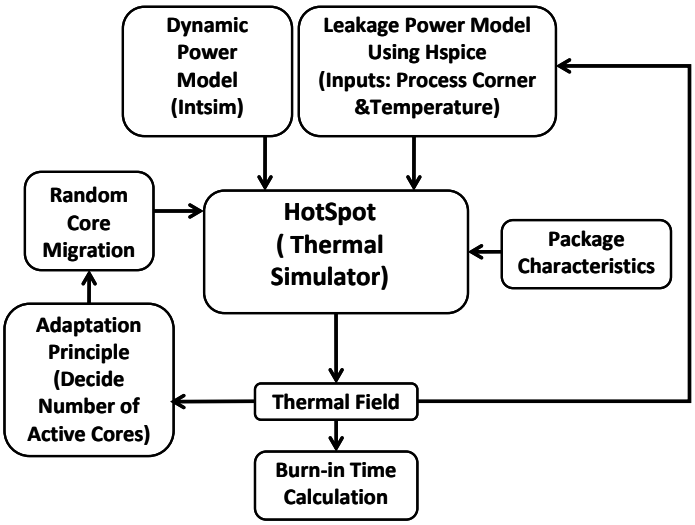



**Figure 3-15: Flow diagram showing implementation of core adaptation policy**

We have explored the application of heat redistribution in burn-in through a technique called "adaptive spatiotemporal power migration (ASTPM) for burn-in of many core chips" [4]. This technique is aimed at reducing the time required to perform the burn in tests while preventing thermal run-away. The ASPTM reduces the burn in test time by upto 20%. In this method we need to adapt the number of simultaneously active

cores depending on the temperature of the chip. The ideal temperature for burn-in test is around 105-110$^{o}$C. In ASPTM, we first turn on all the cores. As the temperature of the chip increases to 110$^{o}$C, a corrective mechanism triggers which reduces the number of active cores by 10%. As the number of active cores reduces, the maximum chip temperature also decreases. In case the temperature again hits the thermal constraint of 110$^{o}$C, we reduce the number of simultaneously active cores again by 10% and so on. This adaptation mechanism has been included in the infrastructure which enables us to device such novel techniques. The results with adaptation of number of cores depending on the process corner are shown in Figure 3-16. It can be seen that the number of adaptation steps for the low $V_t$ corner are more as compared to the nominal $V_t$ corner (which implies that less number of cores are active). Each time the temperature hits 110$^{o}$C, we reduce the number of active cores until the temperature stabilizes at around 105$^{o}$C.

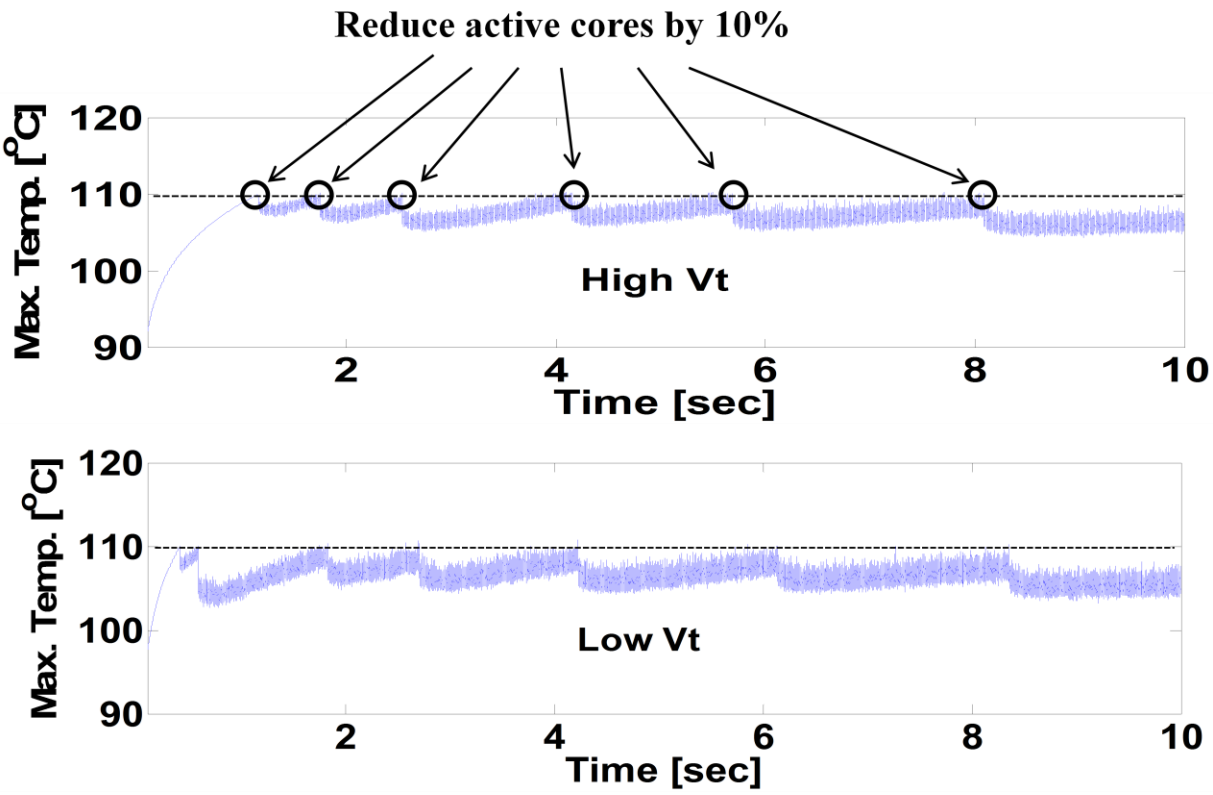



Figure 3-16: Adapting number of active cores

## 3.4 Modeling thermal management policy with constrained migration distance

**Migration distance:**

Whenever we turn a core off, we need some other core on in order to maintain the same throughput. The distance between the core which was turned off and the core that was turned on is called the migration distance. In our setup, we consider the unit of migration distance as "number of hops". Thus, if the core migrated to a core which was directly connected to the core which was turned off, then the migration distance will be '1'. Thus, we can understand that the migration distance will be related to the network topology used to connect all the cores in a chip. If we have a fully connected network, the migration distance will always be 1. But the network will be very complex. Often we have to make the network simple so that the complexity of the network design remains low. In such cases, the migration distance will be different. In our current setup, we assume that the cores are connected using a mesh network as shown in Figure 3-18: . If we assume that the core C01 is turned off and the core C22 is turned on instead, then the minimum migration distance is 3 hops i.e. the state transfer needs to be done between minimum 3 sets of cores.

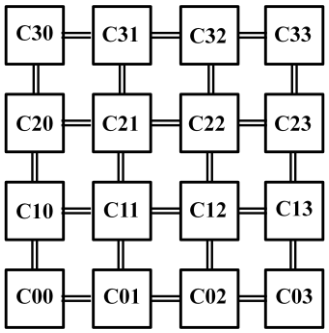



Figure 3-18: Network topology for the chip

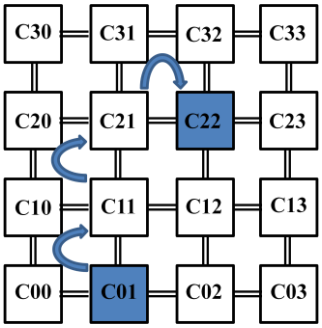



Figure 3-17: Illustration showing migration distance

The number of hops plays an important role in the migration policies. The greater the migration distance, the greater is the cost of migration. For example, if we need to transfer the state of the core to another core, the more the distance, the more will be the time required for this state transfer to happen. The migration distance also has some implications in terms of power. If the migration distance is high, more power is required

to transfer the state of the core to the new location. But often (in some migration policies), we need to have a large migration distance so as to exploit the thermal advantages. Thus we need to make tradeoffs between the thermal advantage and the power-performance cost.

**Thermal management by constraining migration distance:**

Migration of cores is a very good phenomenon from the thermal perspective. But migration causes severe performance degradation if not done optimally. The performance degradation mainly comes from the distance of the older core and the core to which the computation has migrated. In order to address this issue, we try to introduce another feature in the environment. This feature enables us to limit the distance within which the migration can happen. For example, if we constraint the migration distance to "1" (i.e. 1 hop), only the cores on the east, west, north and south can be chosen for migration.

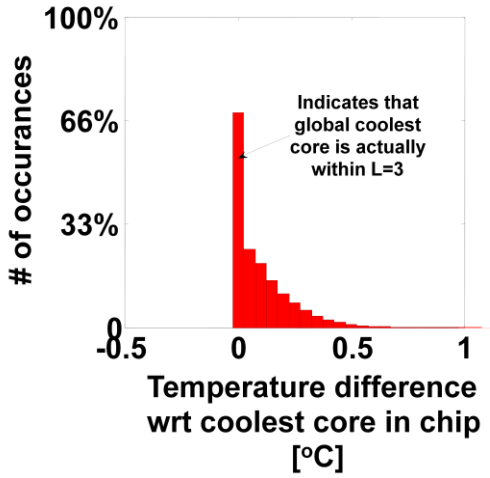



Figure 3-19: Motivation for constraining the migration distance

Another reason to limit the migration distance is: In most cases, under practical workloads, it is highly probable that a core cool enough will be found in the immediate neighborhood of the hot core. This observation is summarized in Figure 3-19. In Figure 3-19, we can observe that around 70% of the times, the coolest core in the entire chip is within 3 hops of the hottest core. At other times, the temperature difference between the coolest core in the chip and the coolest core within a 3 hop distance is very small. So we can constraint the maximum migration distance to 3 hops and still have a very good

thermal performance. At the same time we are reducing the performance degradation by limiting the migration distance. This is another important feature of the modeling setup.

## 3.5 Summary

In this chapter we have discussed the different thermal metrics which we have used to evaluate the thermal performance. We also discuss how the developed thermal modeling infrastructure can be used to model different scenarios such as different number of on-chip sensors, proactive migration, reactive migration, etc. We have discussed how basic thermal management policies such as random, cyclic and global coolest replace can be implemented in the developed framework also how adaptive management methods can be used in case of burn-in. We have presented the concept of migration distance and how it can be used in thermal management policies.

# 4 WEIGHT BASED THERMAL MANAGEMENT

Most of the management techniques which are implemented today are mainly focused on reducing the power when the maximum chip temperature hits a threshold value. Some other thermal management techniques try to redistribute the power across the chip to control the chip temperature. Over the duration of this project, we have analyzed a number of migration techniques which optimize various other metrics besides just the maximum chip temperature such as spatial difference, migration distance, thermal compactness, etc. But in all these migration techniques discussed earlier, the main mechanism was to sample the temperature of the core and take decisions based on this sampled value of the core temperature. Choosing the destination also involved sampling the temperature of the core and then intelligently shifting computation from a hot core to a cooler core. In case of power multiplexing (discussed in chapter 2,3), we assume that a certain percentage of cores is inactive. The computation on the hot cores is then shifted to the inactive cooler cores. But in practice the system may be heavily loaded with a very high percentage (90-100%) of the cores being active at any given time. In such cases, we need to investigate other techniques which can control the thermal field in manycore processors.

In practice different cores run different threads. As the number of cores in a manycore chip is large, there can be potentially a large number of threads running on the chip. But each thread has a different power profile. Some may have lots of memory accesses causing the core to stall and consume less dynamic power, while other threads may consume comparatively large dynamic power making the cores much hotter. We can exploit this variation in the power of the different threads. We can manage the temperature using the “thread swapping” technique which swaps the location of the high temperature and low temperature threads. When we employ thread swapping as a thermal management mechanism, we face an interesting challenge. It is expected that the high power threads will heat up a core much faster as compared to low power threads. Whenever the temperature of the core running a high power thread crosses the predefined threshold, we need to shift the computation to a different core which is cooler than the current core. But while shifting this computation from the hotter core to the cooler core,

we also have to shift the computation from the cooler core to the hotter core. As a result of this bi-directional shifting, the threads running lower power threads are penalized even if the core temperature of such threads is less than the threshold. Thus, techniques which just take into consideration the core temperature unnecessarily penalize the lower power threads.

In order to quantify the performance, we can use another metric called the living time. The living time of a thread is defined as the time for which a thread continues to execute on a given core before being forced to shift computation. If we calculate of all such living time periods, we get the value of average living time for that thread. The performance of a thread will be more if the living time of the thread is more. It is obvious that if the thread suffers more number of migrations, the average living time of the thread will be small and vice versa. Thus, if we consider the case where the hot threads force the low power threads to migrate, we will also see a significant imbalance between the average living times of different threads running on the chip. In order to simulate this scenario, we have considered the global coolest replace policy. The simulation framework we have used is discussed in following section.

**Simulation framework:**

For evaluating the imbalance in number of migrations and living time experienced by different threads running on a manycore chip, we will consider a 64 core chip. Each of the cores is assumed to run a single thread. The floorplan of such a chip is shown in Figure 4-1.

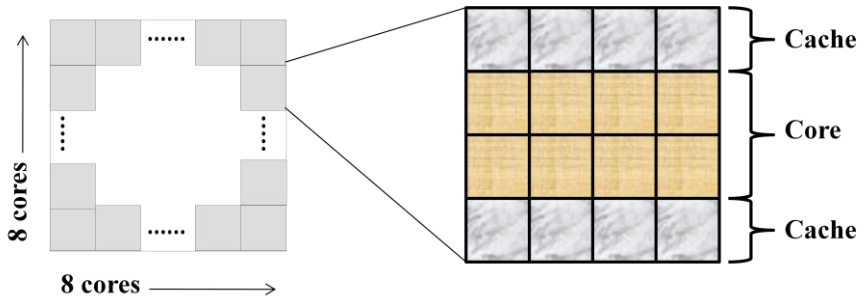



Figure 4-1: Floorplan of manycore chip used for simulations

Each of the 64 cores is further divided into multiple sub-units. There are 2 types of sub-units viz. core and cache. We assume that the thermal sensors are distributed all over the chip at optimal locations. These thermal sensors produce an exact thermal map of the entire manycore chip. The sampling interval of the sensors is assumed to be around 3msec. With this large sampling interval, the accuracy of the sensors will be high and the sensors will be able to project a very accurate thermal image of the chip. In order to emulate the statistical variation in the power of different threads running on the manycore chip, we have generated different power trace files corresponding to different threads. These power trace file have a certain mean value and a certain standard deviation and the distribution is "Gaussian". We generate 64 power trace files, one for each core. Each of these 64 traces is then tied to a certain core in the chip during the simulations. This makes use of the "thread binding" feature of the environment. These trace files are separate for the core units as well as the cache units. The power in the trace file is then distributed amongst the core and cache units appropriately. In real world scenarios, we will have variation in the power of the units within the core. This behavior is simulated by adding a small random power component over the actual power value obtained from the power trace files. Similarly, for cache units, a small power component is added to the base power obtained from the trace file. As random variables are involved in these simulations, we performed (32 runs) Monte-Carlo simulations.

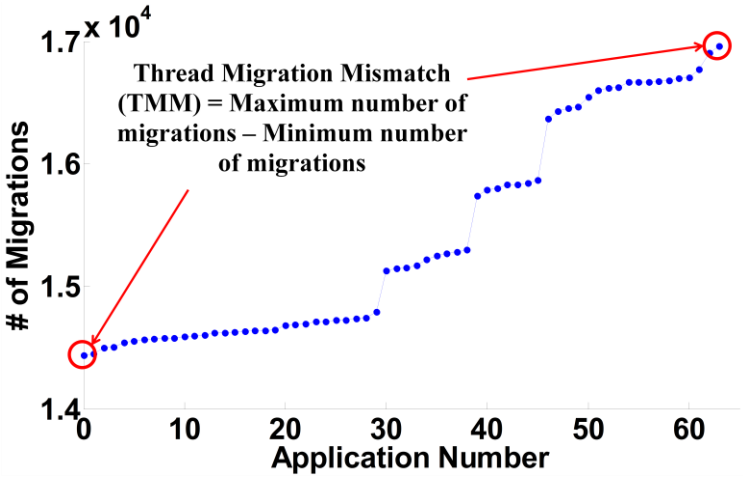



Figure 4-2: Migrations experienced by different applications

The variation in the number of migrations experienced by the different threads is shown in Figure 4-2. The implication of having different number of migrations for different threads is that the threads which have a larger number of migrations, will experience greater performance degradation. This is because, each time we migrate to a new location, we may need to transfer the state of the computation. Moreover, additional power is consumed while shifting the computation from one core. Also, time is required to charge the internal capacitances of the new location which might also consume a lot of time. All these factors cause performance degradation which is proportional to the number of migrations experienced by a thread. We define a new metric called "thread migration mismatch (TMM)" to evaluate the performance overhead caused due to unequal number of migrations. The more the value of the TMM, more will be the performance mismatch between the different threads. Thus, if we desire that all the threads running on the chip be given equal priority in terms of number of migrations, the value of TMM should be as small as possible. The migration difference is calculated as:

$$TMM = \max\left(\#\ of\ migrations\ by\ any\ thread\right) - \min\left(\#\ of\ migrations\ by\ any\ thread\right)$$

Similarly, the variation of the average living time of the different threads is shown in Figure 4-3. The average living time is an important metric of quantifying the performance degradation experienced by a thread. If the living time of a thread is less, then it means that the thread has to suffer larger overhead (in terms of time) to complete the same amount of work.

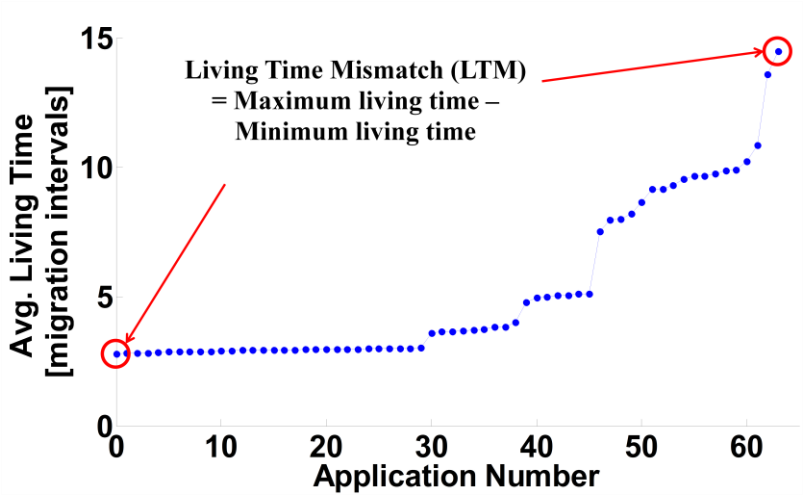



Figure 4-3: Average living time for different applications

As seen in Figure 4-3, most of the threads have a very short living time as compared to the maximum living time of any thread in the chip. Moreover, the difference between maximum and minimum values of living time is large. We define "Living Time Mismatch (LTM)" as another metric which can quantify the performance degradation difference between different threads as a result of the different average living time of the threads. The lower the value of LTM, the less is the performance difference between threads due to average living time mismatch. The LTM can be calculated as:

$$LTM = \max(average\ living\ time\ of\ any\ thread) - \min(average\ living\ of\ any\ thread)$$

In some cases, it is possible that one of the two metrics is more strongly related to the performance degradation. In such cases, we need to have a separation between both the metrics in order to be able to analyze such performance degradation. For example, if we have a system where the number of migrations do not affect the performance much (due to the fact that all cores share a lower level cache or the cores are connected by a fully-connected network), then we need to neglect the effect of TMM (as number of migrations will not cause much performance degradation). From Figure 4-2 and Figure 4-3, we can conclude that some threads in core temperature based management techniques have a very high overhead of migration, while others do not. This causes

difference in the execution time of the applications. So, if we have a scenario where all the threads running in the chip have equal priority in terms of completion, then this priority will not be honored. In fact, some threads will be penalized due to the thermal behavior of other threads.

## 4.1 Weight based management policy (WBMP)

We now discuss a management technique which is "weight based" and enables us to reduce the imbalance in the number of migrations and average living time experienced by different threads, thus making the overhead of thermal management for all the threads similar. We call this new policy "weight based migration policy (WBMP)". The WBMP can be treated as a reactive policy as we take migration decisions only when we have the temperature data available. An important feature of this technique is that we do not take decisions regarding migration based on just the temperature of the core (but also the surroundings of the core). While deciding the source and destination cores for migration, we not only consider the temperature of the cores, but also consider the temperature of the surroundings of the cores.

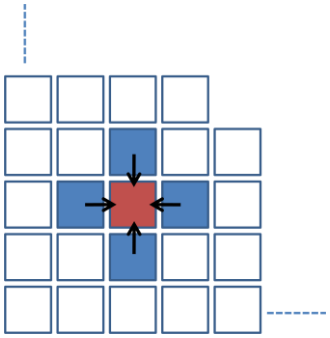


FIGURE:

**Figure 4-4: Core temperature influenced by surrounding core temperatures**

The reason behind using thermal information from the surroundings is: In a manycore chip in highly scaled technologies, the dimensions of the cores are small. As a result of such tight thermal coupling between the cores, the temperature of a core can be influenced by the temperature of the surrounding cores. Thus, even if a given core itself is not the hottest, it is highly probable that we will need to migrate that core if the

surrounding cores are also hot. Similarly, even if a given core is the coolest core in the chip, it may not be a very good destination to shift high power computation if the surrounding cores are very hot. It should be noted that a combination of the current core temperature and the temperature of the surroundings is important in deciding the source and destination for core migrations. Thus, we need to quantify the importance of the core temperature and the temperature of the surroundings in a proper way so as to achieve the correct metric to make migration decisions. We introduce a new metric called "Thermal Weight". The thermal weight is defined as:

$$Thermal\ Weight\ \left(C^{ij}\right) = \{\alpha\ \times maximum\ temperature\ \left(C^{ij}\right)\} + \left\{(1-\alpha) \times \frac{\sum T_{max}^{(cores\ surrounding\ C^{ij})}}{Number\ of\ cores\ surrounding\ C^{ij}}\right\}$$

We call "α" as the "Weight". Thus, thermal weight of a given core is the sum of the temperature of that core scaled with respect to a "weight" α, and the average of the maximum surrounding temperatures scaled with respect to (1-α). Note that for calculating the thermal weight, we use the average of the maximum temperatures of the temperatures. For example, the thermal weight of core $C_{12}$ shown in Figure 4-5 is calculated as follows (assuming α = 1.0).

| | | | |
|---|---|---|---|
| C30 | C31 | C32 | C33 |
| C20 | C21 | C22 | C23 |
| C10 | C11 | C12 | C13 |
| C00 | C01 | C02 | C03 |

**Floorplan**

| | | | |
|---|---|---|---|
| 65 | 61 | 55 | 66 |
| 68 | 75 | 73 | 71 |
| 58 | 56 | 62 | 65 |
| 76 | 69 | 64 | 63 |

**Maximum temperatures of cores**

**Figure 4-5: Calculating thermal weight**

Thermal weight of core $C_{12}$ = {1.0 x 62} + {(1-1.0) x $\frac{56+64+65+73}{4}$ } = 62

Here, the core $C_{12}$ is surrounded by cores $C_{11}$, $C_{02}$, $C_{13}$ and $C_{22}$. We consider only the cores sharing a boundary with the core under consideration as the surrounding cores. This is because only the cores sharing a boundary with the given core will have maximum influence on the temperature of the given core. We now monitor this "thermal weight" in order to make decisions about the source and destination of migration. If the value of α is 1.0, it means that we are not considering the information about the surroundings. Thus, when α =1.0, the weight based policy behaves exactly as the global coolest replace policy. The other extreme case is when the value of α is 0.0. When α = 0.0, we do not consider the temperature of the core at all. Instead, we consider just the temperature of the surroundings of the core to make decisions about the source and destination of migration. The intermediate values of α represent different cases where the temperature of the core itself is scaled according to the value of α.

### 4.1.1 Implementing WBMP in current environment

For implementing the WBMP in the current setup, we first need to enable thread mapping. This technique is discussed in Chapter 2. When we shift computation to another core, we actually swap the threads running on the two cores under consideration. This is called thread swapping. In order to implement this feature, we first need the power profile of each of the threads which are tied to the cores. We setup these thread power profile files during the thread mapping step. We also need to keep a record of which thread is mapped to which core. This mapping information is stored in an internal array called "core_thread_mapping". When we want to swap the threads, we can change this mapping which will emulate the thread swapping behavior. The pseudo-code for thread mapping and thread swapping is shown in Figure 8-10 (in Appendix).

## 4.2 Evaluation of WBMP

The most important metric which needs to be monitored is the maximum temperature in the chip. This metric is first monitored to ensure that the thermal design point constraint is not violated. The main point which separates WBMP from the conventional thermal management policies is the attention it pays to the performance metrics such as total migrations for a thread, the average living time for a thread, the TMM and the LTM. We analyze the effect of varying the weight 'α' on the thermal (maximum chip temperature, spatial difference) as well as the overhead (TMM, LTM) parameters.

## 4.3 WBMP comparison Results

**Thermal parameters:**

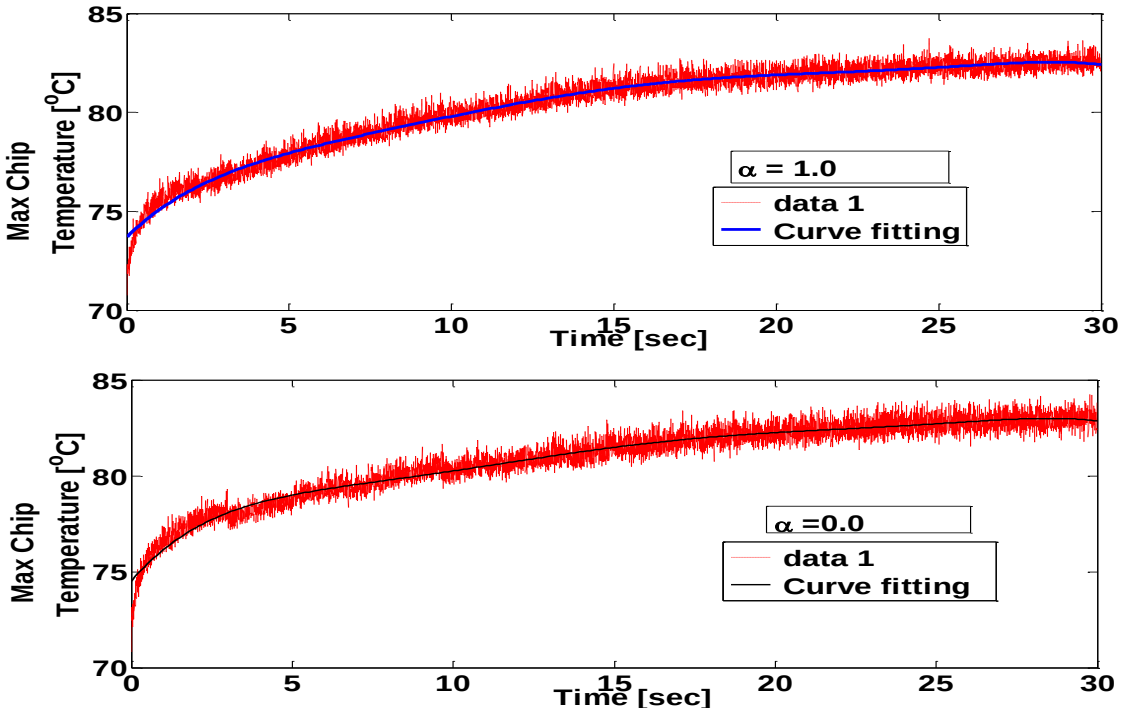



Figure 4-6: Maximum chip temperature vs. time for different values of α

We first evaluate if the WBMP can keep the maximum chip temperature under control. The analysis of the maximum temperature for all the values of α was done. In all the simulations the maximum chip temperature remained under the required limit. The maximum chip temperature for WBMP for α = 1.0 and α =0.0 is shown in Figure 4-6. In both cases i.e. α = 1.0 (global coolest replace) and α = 0.0 (extreme case of WBMP), we observe that the maximum chip temperature is maintained below 85°C at all times. Thermal profiles for other values of α are also very similar.

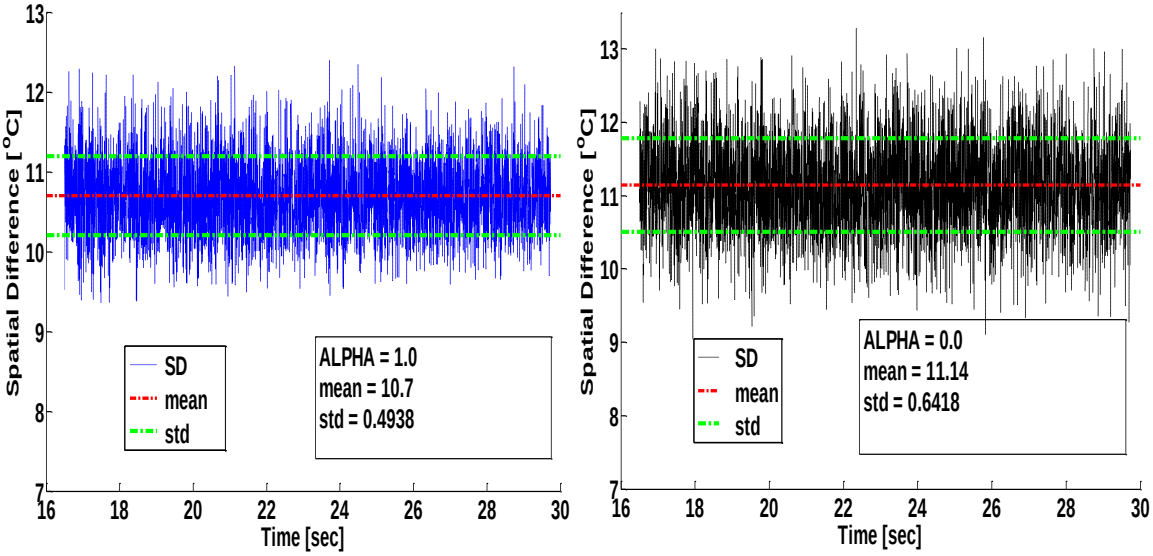



Figure 4-7: Spatial difference comparison

As the maximum chip temperature profile of the two extreme cases is similar, the other thermal metrics are also similar in nature. The Figure 4-7 shows comparison of the spatial difference of the two policies with the statistical parameters. It is observed that the mean spatial difference (SD) is similar in both cases. Also the standard deviation of SD is also similar. Note that the spatial difference is compared after the temperature of the chip has stabilized. From the above observations, we can conclude that the temperature profile of a chip is not severely affected by changing the value of α. (The intermediate values of α also produce very similar thermal profiles as the worst cases). The other thermal metrics such as temporal difference, temporal gradient are also very similar to the conventional global coolest replace policy.

**Performance overhead parameters:**

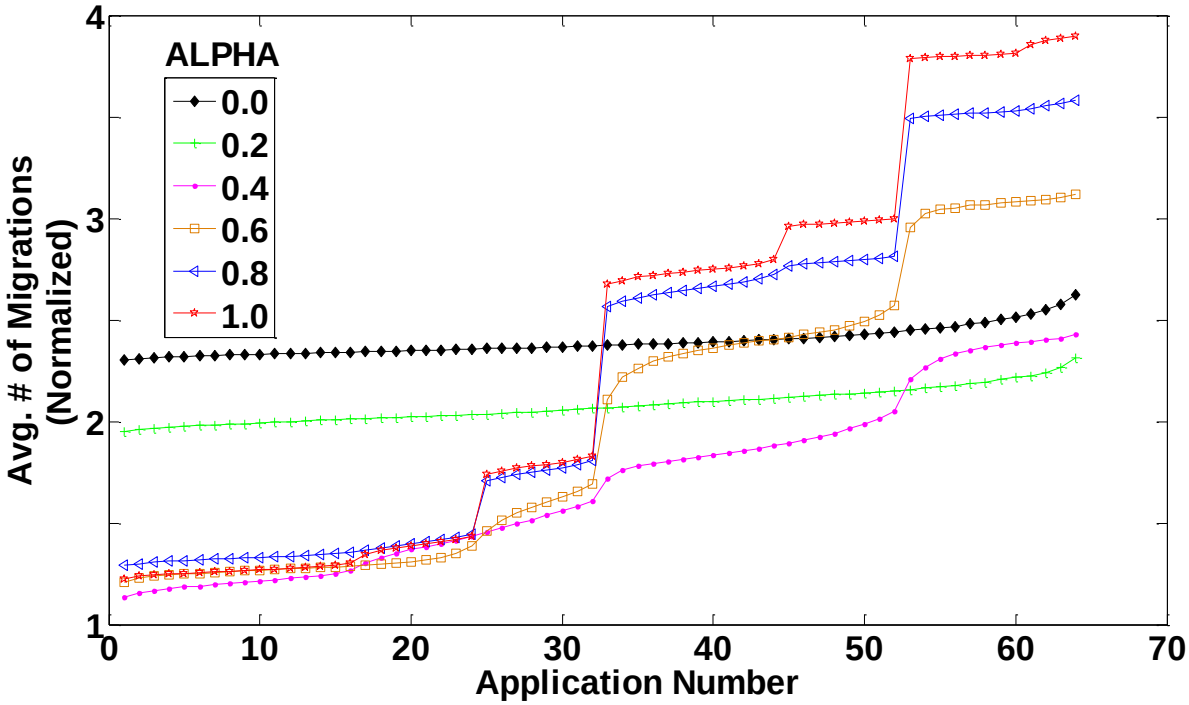



Figure 4-8: Comparison of average number of migrations

Figure 4-8 shows the distribution of migrations for all the threads running on the chip for different values of α. In Figure 4-8 we have considered the average value of the number of migrations for each thread. This average value is calculated using all the monte-carlo simulations run for that specific value of α. As the value of α decreases, the number of migrations experienced by the threads tend to become similar. For high values of α, we observe an imbalance between number of migrations experienced by different threads. But in order to have same performance for all threads, it is essential that all threads experience similar number of migrations. This can be achieved using smaller values of α.

Figure 4-9 shows the average, maximum and minimum number of migrations experienced by all the threads running on the chip for 2 different values of α. Figure 4-9 helps to statistically prove the results. Figure 4-10 shows the number of migrations experienced by the different threads as a function of the thread power. It is obvious from the graph that the high power and low power threads suffer the most number of migrations for high values of α, while the curve becomes flat for smaller values of α. This corroborates the results in Figure 4-8 and Figure 4-9.

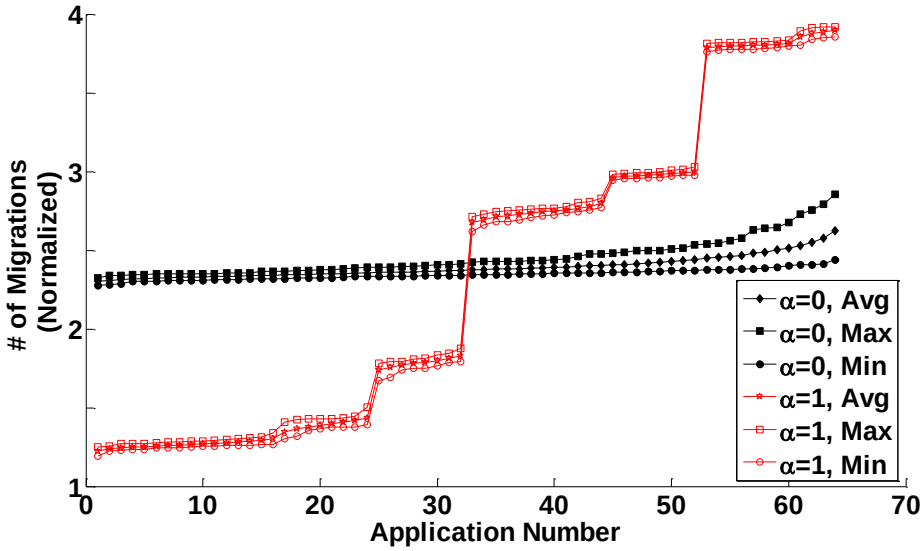



Figure 4-9: Minimum, maximum and average number of migrations

As the number of migrations experienced by the different threads becomes uniform for small values of α, it is obvious that the TMM value should also decrease with decrease in α (as TMM represents the difference between maximum and minimum number of migrations between different threads). This variation of TMM with respect to α is shown in Figure 4-11. From Figure 4-11Figure 4-12: , it is observed that the average number of migrations experienced by all the threads in total has an optimum value for α = 0.4. Thus, according to the metric that we need to optimize, we need to adjust the value of α accordingly.

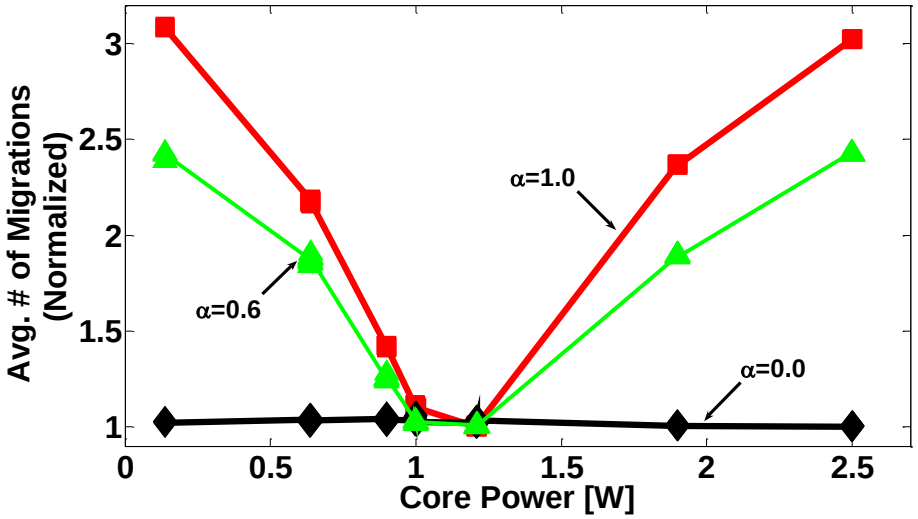



Figure 4-10: Variation of number of migrations with application power

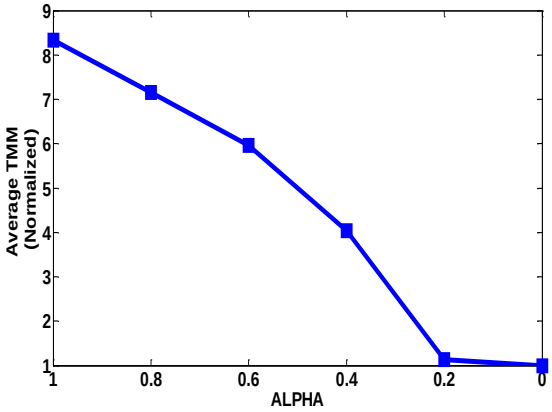



Figure 4-11: Variation of TMM with respect to α

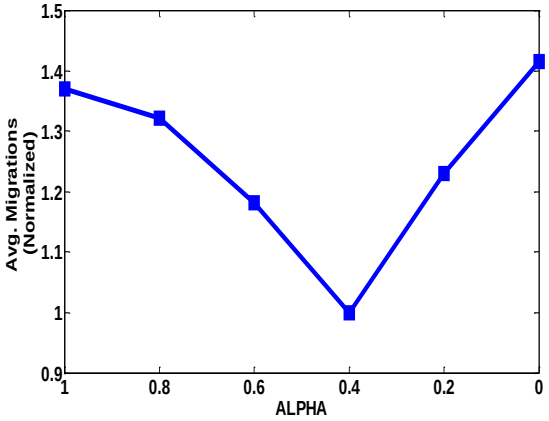



Figure 4-12: Variation of average number of migrations with α

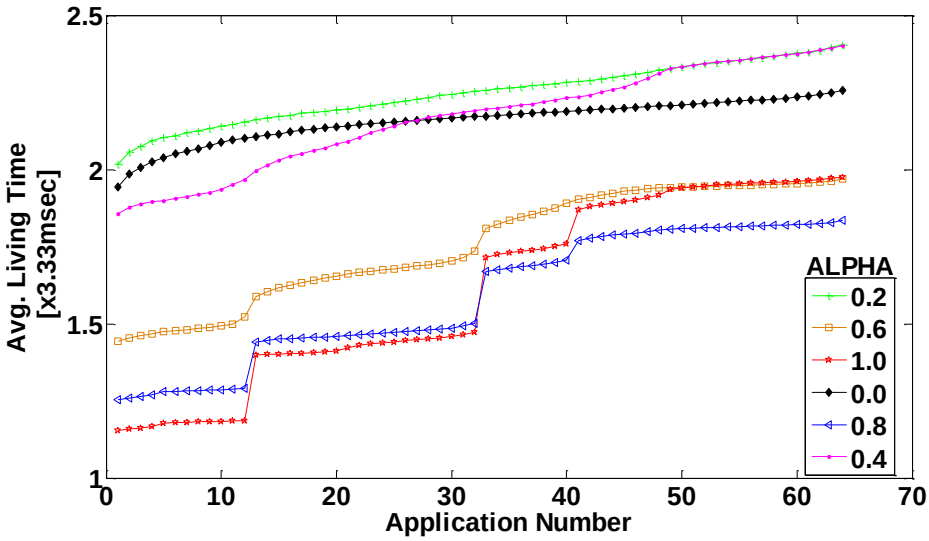



Figure 4-13: Variation of average living time

The other metrics which we use to quantify the performance of the threads are living time and LTM. The threads will have a higher performance if the average living time of the threads is higher. Also, if we want all threads to have same performance, have a similar value of average living time is important. Figure 4-13 and Figure 4-14 show the variation of the living time of different threads. We observe a similar trend in case of living time too i.e. the average living time of different threads becomes similar for small values of α. As a result of this, the LTM values are also small for smaller values of α, which are shown in Figure 4-16. An interesting observation regarding the average living time of threads shown in Figure 4-16**:** is that the value of average living time saturates at

α=0.4 i.e. even if we decrease the value of α from 0.4 to 0.0, we do not have a very significant increase in the average living time of the threads.

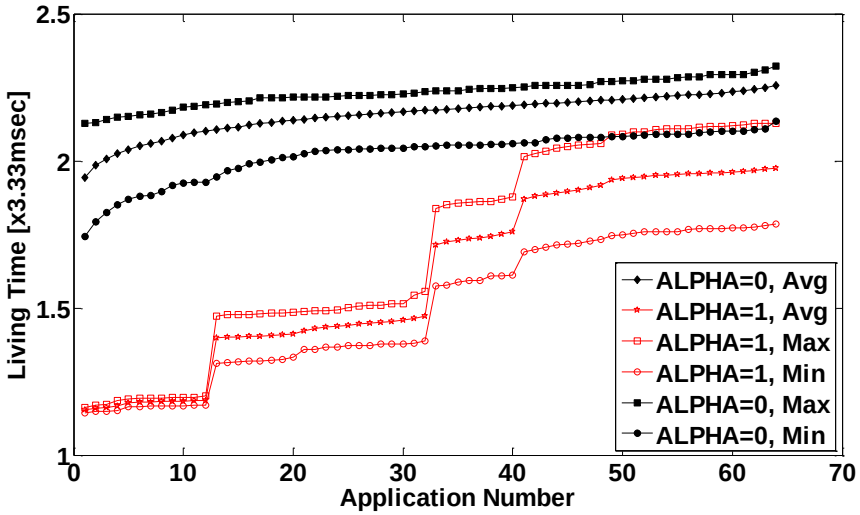



Figure 4-14: Minimum, maximum and average living time curves

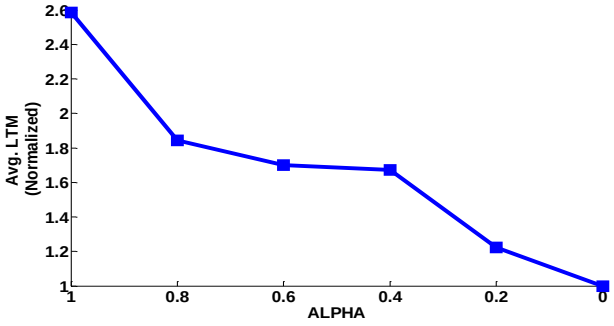



Figure 4-16: Variation of LTM with respect to α

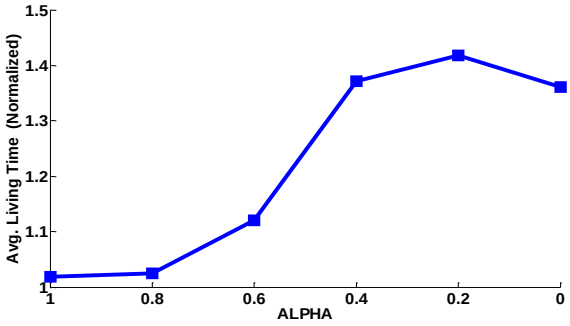



Figure 4-15: Variation of average living time

**Correlation between number of migrations and living time:**

We have observed that as the number of migrations increases, the average living time of the threads decrease i.e. there exists a certain correlation between these parameters. This correlation between total number of migrations and the corresponding average living period is shown in Figure 4-17. The nature of this correlation remains constant even for different values of α.

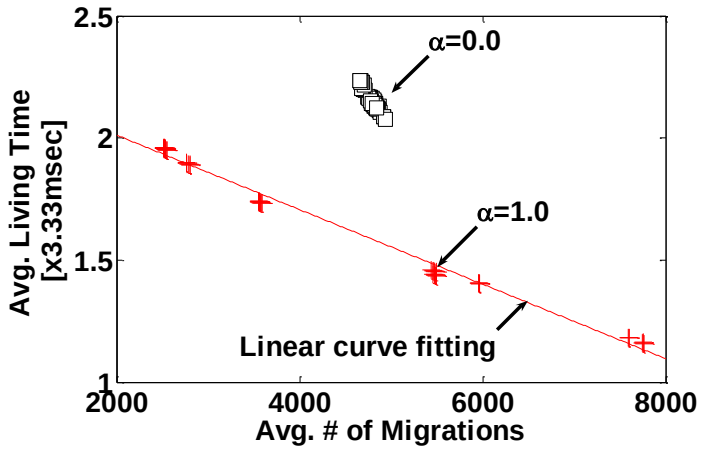



Figure 4-17: Correlation plot between living time and number of migrations

## 4.4 Implementing WBMP for SPEC2006 benchmark traces

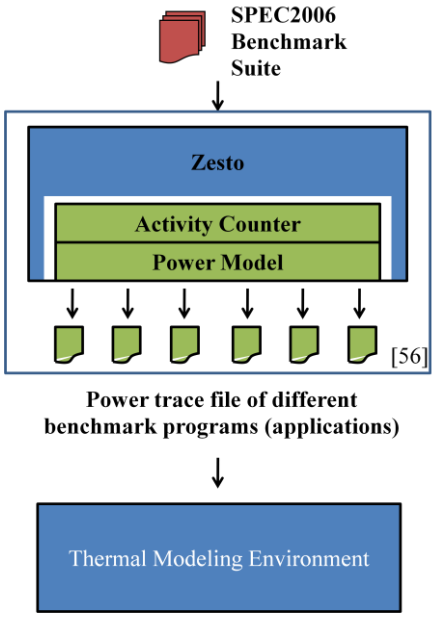



Figure 4-18: Framework for generating SPEC2006 benchmark traces

In order to implement the WBMP for SPEC2006 benchmark traces, the framework shown in Figure 4-18 can be used. The basic environment for modeling the thermal behavior remains same as discussed earlier. The additional part which is included in this framework is the architectural simulator Zesto[55]. Zesto is a cycle perfect architecture simulator which can model the x86 architecture. In order to get power traces

of real world programs, we need to add additional components to the architecture simulator. The basic components that need to be added to Zesto are the activity counters and the power model. The activity counters are responsible for counting the actual number of times the block is activated. The dynamic power consumed by the block is directly proportional to the activity of the block. This relation is based on the dynamic power equation discussed in chapter 1. We have integrated a C++ module to Zesto which can measure the activity of each of the blocks in the architecture. Separate counters are necessary for the cache and core blocks as we need to have separate power trace files for cache and core in the thermal modeling environment.

The second component added is the power model. The power model is responsible for converting the activity count to the corresponding power value. A power value is generated every 100us using this framework. We can then sample the power values as required at a later stage. Once we have the power trace files for the applications in the benchmark, we create multiple instances of these power traces. These trace files are then tied to the different cores in the chip as discussed in chapter (thread mapping). Then, WBMP is enabled and simulations are performed for different values of α. For obtaining the power traces, the "astar-biglakes", "astar-rivers", "bzip2-chicken", "bzip2-combined" and "bzip2-liberty" programs frpm the SPEC2006 benchmark suite were used.

## 4.5 Simulation results for the SPEC2006 benchmark traces

**Thermal parameters:**

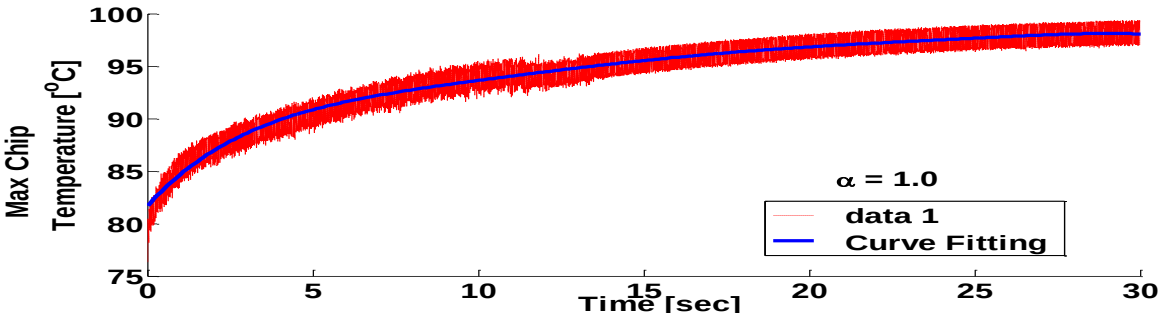



Figure 4-19: Maximum chip temperature variation (α=1.0)

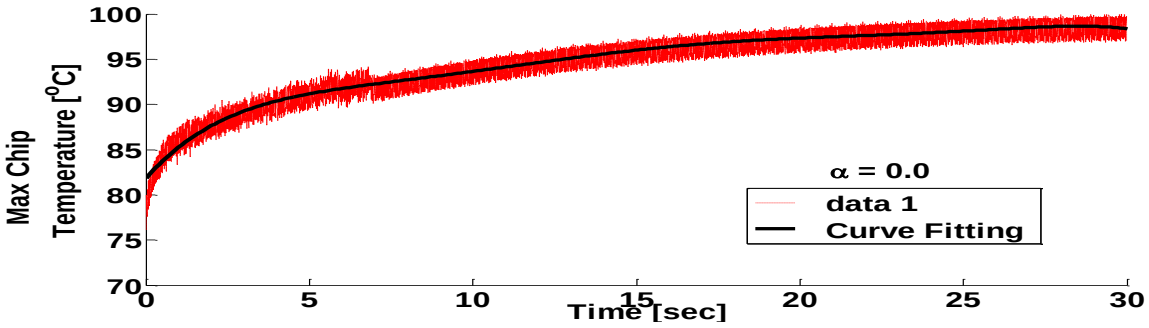



Figure 4-20: Maximum chip temperature variation (α=0.0)

We observe that the global coolest replace and WBMP have similar maximum chip temperature profiles even for the SPEC2006 traces. This is shown in Figure 4-19 and Figure 4-20.

**Performance overhead parameters:**

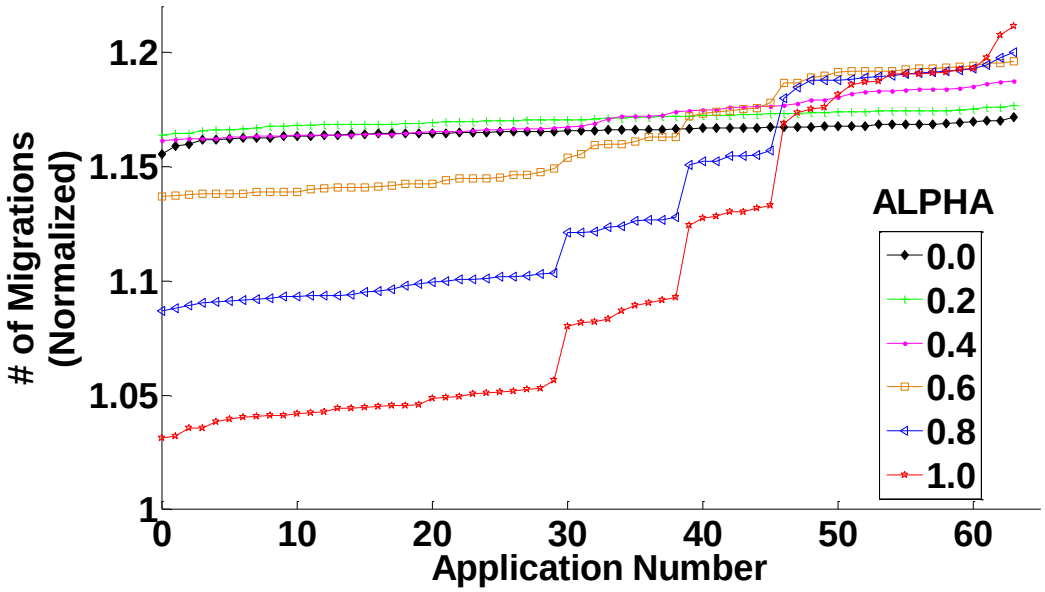



Figure 4-21: Variation of number of migrations

Figure 4-21 shows the average number of migrations for all the applications running on the chip. We can see a similar trend as compared to the simulations carried out with random power traces. Similarly, the nature of the living time curves for different values of α follow a similar trend. This is shown in Figure 4-22.

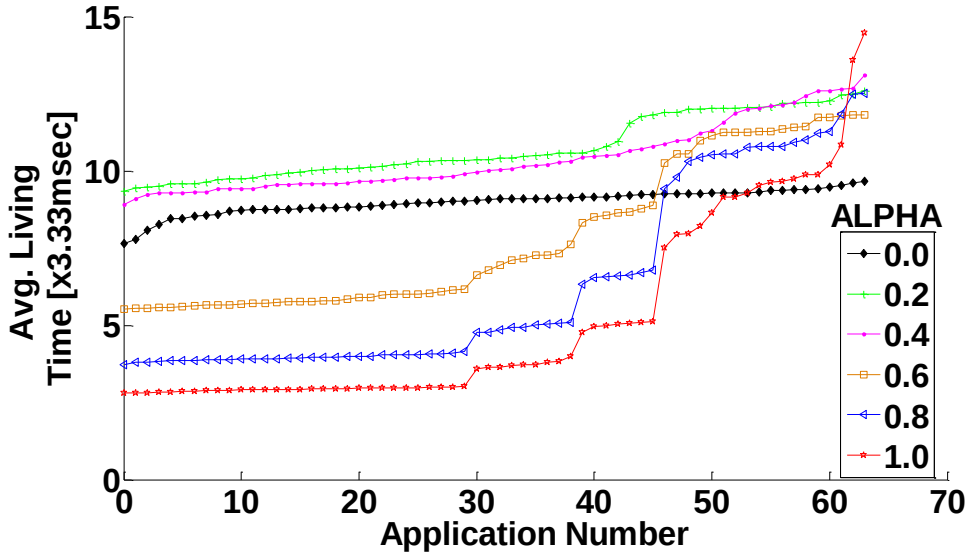



Figure 4-22: Variation of average living time

Figure 4-23:  and Figure 4-23 show the variation of TMM and LTM with respect to α. Lower values of α give lower values of TMM and LTM which indicate that all the threads experience similar number of migrations and average living time.

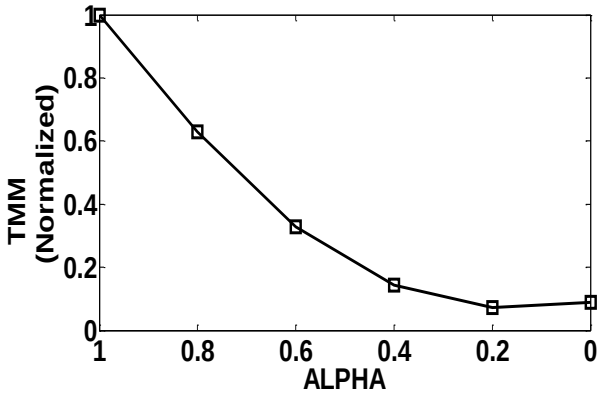



Figure 4-23: Variation of TMM with respect to α

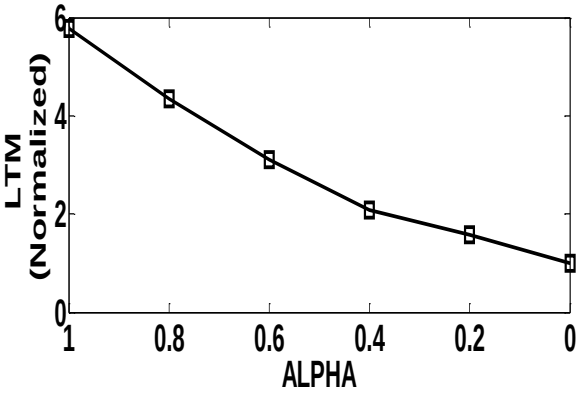



Figure 4-24: Variation of LTM with respect to α

An important difference between random trace simulations and real world trace simulations is the absolute value of the migrations experienced. The real world traces from the 2006SPEC benchmark suits have a higher power as compared to the random simulations. As a result, the absolute number of migrations increases by a factor of 10.

## 4.6 Extension of WBMP

Changing the value of α has different effects on the different performance parameters of the system. For example, reducing the value of α from 1.0 to 0.0 makes the number of migrations experienced by the threads uniform i.e. decreasing the value of α from 1.0 to 0.0 brings in a notion of performance fairness to the system. This property of the manycore system can be used in order to manipulate how the threads are executed in the manycore chip. If we have a scenario where all the threads running on the chip have equal priority and need to be completed at approximately the same time, we can modify the value of α to a lower absolute value thus increasing the performance balance between threads Thus, if we want threads to experience very less migrations, then we can tune the value of α accordingly. In the current setup, the weight 'α' is used to scale only the physical temperature value of the core and the surrounding. We then calculate the 'thermal weight' of each of the cores in the chip. As we are using only the temperature is this calculation, the thermal weight matrix of the cores is a function of the core temperatures i.e.

$$Thermal\ Weight = f\left(T_{C^{ij}}\right)$$

In case if we have deadline driven applications running on the chip. These time critical applications need to be finished as early as possible. In order to enable this scenario, we need to ensure that such time critical applications do not suffer large number of migrations i.e. the time critical applications should suffer less number of migrations as compared to the other applications. If we consider the case where α = 1.0, we observe that some applications to have significantly less number of migrations as compared to others. So, we can selectively make the time critical applications suffer less number of migrations as compared to other applications by using higher values of α. But in addition to the physical information, we also need to assign a priority to such threads so that we can selectively push such threads to the "less migrations" zone in the migration curve. Thus, the 'thermal weight' of such threads not only contains information about the

physical core temperatures, but also about the time criticality of the thread (i.e. the priority). We will now call the 'thermal weight' as 'logical weight'. Thus,

$$Logical\ Weight = f(T_{C^{ij}}, Thread\ priority)$$

We have now included some other information in addition to the physical core temperature information. Thus, we can now calculate the 'logical weight' of each core and then sort the array of 'logical weights' and take decisions regarding migrations. In this manner, we can selectively force time critical threads to suffer less number of migrations. This concept of 'logical weight' can be further extended to include other factors which can be used for deciding the performance of a thread. For e.g. if a certain thread has to suffer a very high performance degradation if the average living time value of that thread is low, then we can incorporate the average living time value in the calculation of 'logical weight' for all the threads. By including this information in the 'logical weight' we can force threads having high performance degradation with respect to living time value to have a higher value of average living time. In this case,

$$Logical\ Weight = f(Average\ Living\ Time)$$

In certain cases, we may need to include more information related to the physical characteristics of the chip. For example, if the core dimensions are small, then more the gradient between a core and its neighbor more will be the influence on temperature. Thus, we can also include gradient information while calculating the logical weight of the cores. In such a case, the logical weight will be a function of the temperature gradients (instead of the average of the maximum surrounding core temperatures as we have considered currently). i.e.

$$Logical\ Weight = f(Temperature\ gradients\ between\ adjacent\ cores)$$

Another interesting parameter which can be included while calculating the 'logical weight' of the cores is the thread or application characteristics. For e.g. if we

consider heterogeneous (asymmetric) manycore systems, some cores are better suited to execute single threaded applications. If there is a need to migrate such threads to simpler cores, we may end up having severe performance loss. Other characteristics of threads such as instruction density, etc can also be included for calculating the logical weight of the threads. If we consider another case where two adjacent cores are executing threads and the execution is closely matched (or the effective throughput obtained is more as compared to other combinations of cores), then we have to consider such information when we take decisions regarding migrating the threads to other locations. Another example is if some of the cores are running threads for scientific computing and the locality is very high, then the migration cost of such threads is high. In such cases, we can include such thread characteristics and take appropriate migration decisions on a thread to thread basis.

In this work we have considered a wide distribution in the power of different threads running on the chip. This wide variation in the power of the threads can be filtered to achieve the required performance of the threads. We can change the value of α and change the variation in the performance of different threads. In WBMP we exploit the variation in power of different threads. We also need to explore the effectiveness of the WBMP if the power variation is very low. In such cases, it may be possible that changing the value of α does not give significant benefit.

By including logical information as well as other physical information and using the concept of logical weight, we can also design adaptive techniques which adapt to different requirements at different time intervals. Thus, this concept of 'weight' can be extended to include other information and this can help us optimize the system for the required parameters. In general, the logical weight can be considered to be a function of physical parameters (such as absolute core temperature, core temperature gradients, core size, cooling solution information, etc) as well as logical parameters (such as thread characteristics, relation between migration and performance degradation, thread power profile, etc) i.e.

$$Logical\ Weight = f(Physical\ parameters, Logical\ parameters)$$

## 4.7 Summary

The weight based management policy can be used to introduce a notion of fairness amongst the threads running on the manycore chip. As the value of the "weight (α)" decreases, all the threads running on the chip experience similar number of migrations and living time values. We can also adaptively configure the WBMP to serve the required performance for different threads for e.g. the value of α can be modified according to the time criticality of the threads running on the chip. We can also extend the concept of 'weight' to include other logical as well as physical information in order to make the system more efficient.

# 5 MODELING TOOL

## 5.1 Overview and flow

In our effort to understand the thermal behavior, we believe that percolating knowledge gained from this research to the current students which will design future generation processors is most important.

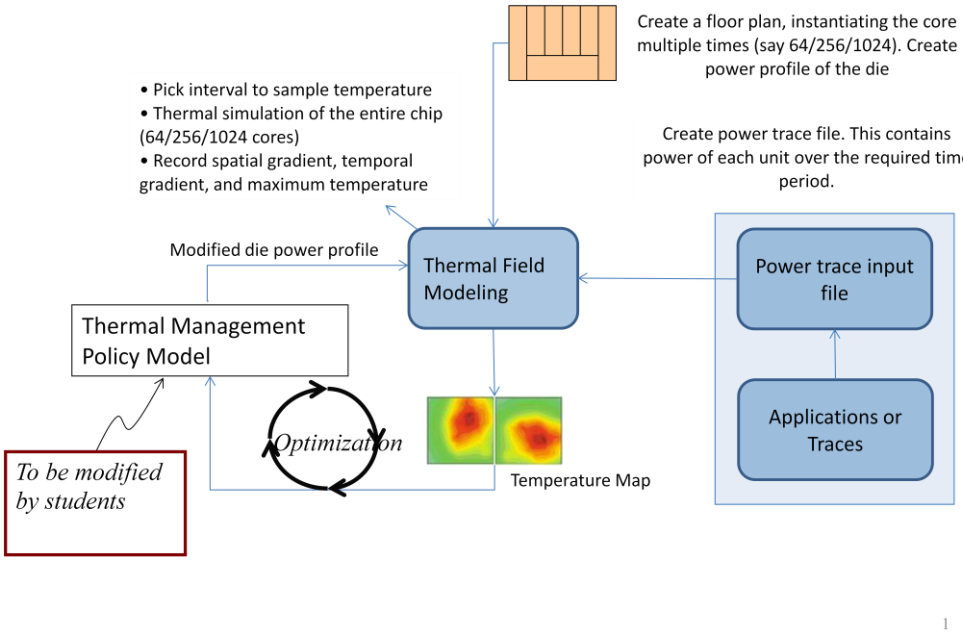



**Figure 5-1: Flow diagram of the GUI based modeling tool**

As a step towards realizing this goal, we have built a tool which gives an idea to the students regarding how thermal modeling in manycore processors can be done as well as gives a base on which students can construct innovative solutions to enable thermal management in manycore processors. All the components which we have used in our research viz. HotSpot [12], IntSim [8], generating power trace files, generating floorplan files are included in this tool. The basic flow diagram of this tool is shown in Figure 5-1.

In order to make it easier for the users to use this tool, we have created a GUI which enables the user to run the tool by just clicking certain buttons on the GUI. The main components of the tool can be listed as follows:

1. Graphical user interface (developed in Python)
2. Script (in Perl) to create the required floorplan file
3. Script (in Perl) to create the required power trace file
4. HotSpot core (which gives the temperature map as the output)
5. Post-processing scripts (in Perl) which analyze the temperature map and produce Matlab readable (.m) files as an output. These .m files can then be imported into Matlab to plot different graphs.

## 5.2 Graphical user interface

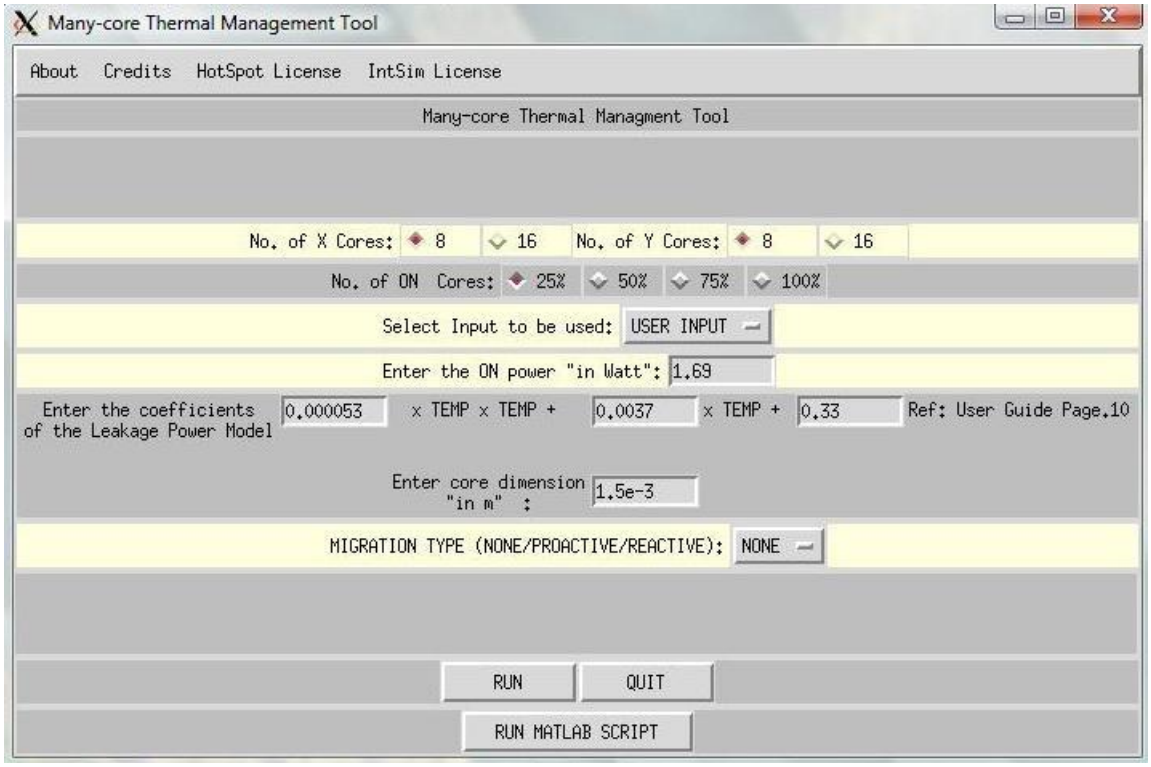



Figure 5-2 Graphical User Interface

As described earlier, the Graphical user interface (GUI) provides a simple interface with the thermal modeling tool. The GUI is developed using Python. The GUI is shown in Figure 5-2. The main fields which are present in the GUI are as follows:

**Number of cores in the chip:**

No. of X cores, No. of Y cores: In this version of the tool, we have used to the floorplan structure shown in Figure 5-3. The "No. of X Cores" is the number of cores in chip which are in the X direction while "No. of Y Cores" is the number of cores in the chip which are in the Y direction.

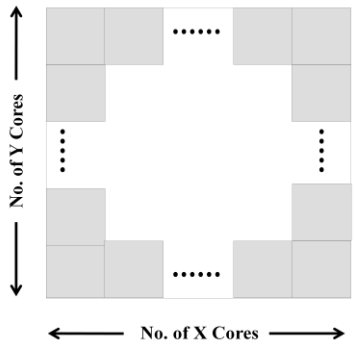



Figure 5-3: Floorplan of the manycore chip used in the modeling tool

In the current version of the tool, we provide 2 options to select the number of cores in the chip. The total number of cores in the chip can either be 8x8 (i.e. 8 cores in X direction and 8 cores in Y direction to give a total of 64 cores) or 16x16 (i.e. 16 cores in X direction and 16 cores in the Y direction to give a total of 256 cores) cores. The number of cores which are selected is then used to run a script (called generate_flp.pl) which generates the required HotSpot compatible floorplan of the chip. We can easily modify the number of cores we want to use by modifying the script used to generate the floorplan. One thing that needs to be noted is that the power trace file also changes if the floorplan file changes. This is because HotSpot requires that the units specified in the floorplan file match the units specified in the power trace file. Thus, whenever we modify the floorplan file, we must also modify the power trace file accordingly.

**Number of active cores:**

The temperature of the chip changes according to the number of cores that are active at any time. The number of cores that are active at any given time in the simulation can be controlled by the "Number of ON cores" field in the GUI. The options that are available are 25%, 50%, 75% and 100%. The absolute value of the active cores depends on the total number of cores which are in the chip. For example, if the total number of cores in the chip is 64 and the number of active cores is 25% then the actual number of cores that are active is 16. Similarly, if the number of cores in the chip is 256 and 50% of them are active, the actual number of active cores is 128, and so on.

**Power input selection:**

In order to get the temperature profile of the chip over time, the user needs to specify the power or each unit over time. This power is the dynamic power dissipated by each of the units specified in the floorplan file. The dynamic power of a circuit can be calculated using multiple ways. If we have the actual power numbers from real systems, we can directly use the numbers. If we do not have the exact power values, then we can estimate the dynamic power of the chip. One such tool which models the power of a chip is called IntSim. In our tool, we give the option of using specifying the dynamic power using either IntSim or directly from the user.

In case the user wants to specify the value of dynamic power, the user needs to select the "User Input" in the "Select input to be used" option provided on the GUI. The value of dynamic power can then directly be input to the tool. The other option is to use IntSim to calculate the dynamic power of each of the cores. In case we want to use IntSim to calculate the dynamic power of the core, we need to specify all the internal technology and circuit details. IntSim needs 4 types of parameters: system, device technology, interconnect and package parameters and design parameters. The interfaces for specifying all these parameters are shown in Figure 8-11, Figure 8-12, Figure 8-13 and Figure 8-14 (in Appendix).

## 5.3 Case study

Problem statement: Model a 64 core manycore chip. The manycore chip is a homogeneous multiprocessor chip. Assume the dimensions of each core of the chip to be 1.5mm. The dynamic power dissipated by each core is 1.2W. Assume that the leakage power for the target technology is independent of the operating temperature, but a constant leakage power of 0.2W is dissipated in each core. Evaluate how the maximum temperature of the chip varies with respect to time for 75% of the cores active at all times. Assume that no migration policy is implemented.

**Solution:**

1. It is given that the manycore chip consists of 64 cores. We will assume that 8 cores are in X direction and 8 cores are in the Y direction. To enter this information, first launch the GUI. Then select the "8" as the "No. of X Cores" value. Similarly select "8" to the value of "No. of Y Cores".
2. As 75% of the total cores are given to be active during the simulation period, we select the "75%" radio-button for the "No. of ON Cores" field.
3. In this problem, the power dissipated by the cores is directly given to us. Hence, we will select "user input" from the selection menu of the "Select input to be used" field.
4. The dynamic power is then entered in the "Enter ON power box" i.e. we enter 1.2 in the box in front of "Enter ON power" field.
5. In the problem statement, it is given that we need to add a constant leakage power to each core. In the GUI, we generally enter the equation of the leakage power – temperature interaction curve. The equation is of the form:

$$Leakage\ Power = \{A \times Temperature^2\} + \{B \times Temperature\} + C$$

   In the GUI, we enter the values of A, B and C in the given boxes. As we need to add a constant power value, we will enter A = 0, B = 0 and C = required constant value i.e. C = 0.2.
6. The core dimension needs to be entered in the "Enter core dimension" field box.

7. Here, we have to evaluate the temperature without implementing any migration technique. So we will choose “None” from menu for the “Migration Type” field.
8. We then run the simulation and get the required temperature trace files.

   The GUI with all the required fields filled out with appropriate values is shown in Figure 5-4.

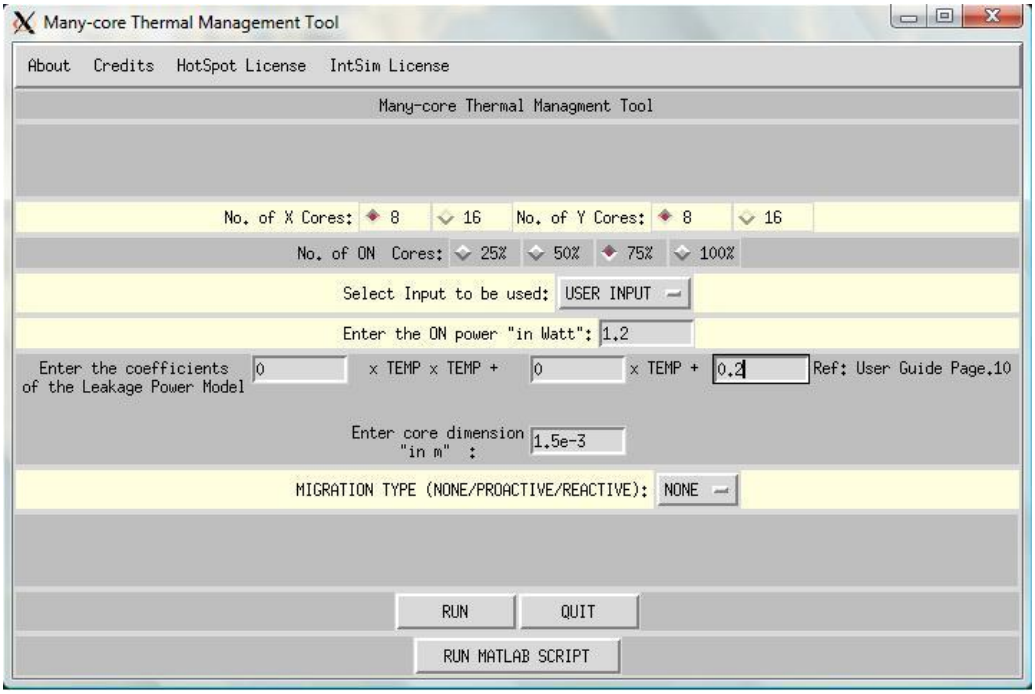



**Figure 5-4: GUI showing values entered for the case study example**

## 5.4 Expanding the modeling tool

The current version of the modeling tool contains features required for modeling the thermal behavior of the future generation manycore processors. In order to completely understand the working of the complete system with implications of the thermal limitations on the performance of the system, we need to add many more components to this infrastructure. Another important point which needs to be evaluated is the effectiveness of the thermal management policies on real workloads. For different domains (such as embedded systems, general purpose CPUs, graphics processing units, etc) the workloads are very different. Thus, we need to include much more flexibility in

the current environment such that all the other domains can make use of this modeling framework.

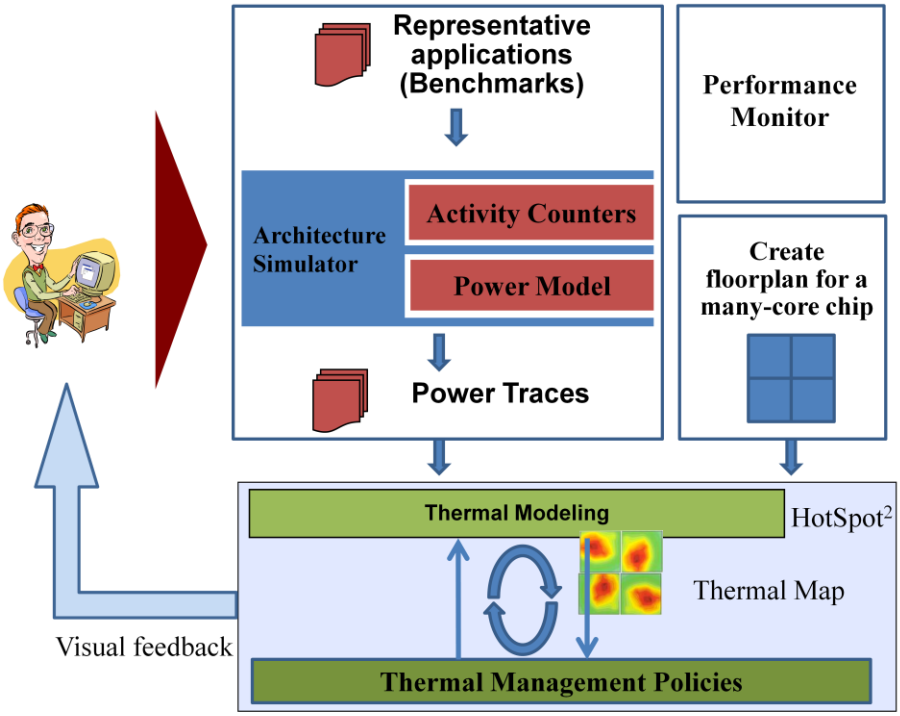



Figure 5-5: Enhanced version of current modeling setup

For including real world power traces in this simulation framework, we envision the integration of an architecture simulator with the current setup. This architecture simulator like Zesto [55] can simulate real world benchmarks and then provide the power profile of these benchmarks which in turn can be plugged in the current setup. Modifications may be required to the architecture simulator in order to calculate the power of the entire design. The flow diagram with an architecture simulator integrated with the current setup may look as shown in Figure 5-5.

## 5.5 Summary

The GUI based tool is a very basic tool used to model the thermal fields in manycore processors. This tool needs to be modified by students in order to develop innovative thermal management techniques. The GUI based tool provides a base on which the users can easily add a large number of features so as to model the target manycore processor.

# 6 CONCLUSION

In this work, we have developed a framework to simulate the thermal field in future generation manycore processors. The framework is constructed around the HotSpot thermal simulator developed by university of Virginia. Numerous additional features such as capabilities to change the number of thermal sensors, the location of thermal sensors, the migration policies, the temperature- leakage power interaction, inter-die and intra-die variation have been to the infrastructure. These features enable us to model different scenarios which may exist in the manycore chips. We have introduced different migration policies such as random migration, cyclic migration and global coolest replace as the basic thermal management policies.

The main contribution of this work is setting up the modeling environment so that we can use it to evaluate different thermal management policies under different physical scenarios. We have used this framework to propose a weight based management policy. This new policy enables us to include “fairness” in the execution of the different threads running on the chip. We can change the value of the “weight” so that we can achieve different performance for the threads running on the chip. First, we statistically evaluate the weight based policy using dummy power traces and then use SPEC2006 benchmark power traces to validate the policy. We also evaluate the effect of number of active cores on the weight based policy.

Finally, we discuss our efforts to transfer the knowledge gained from this research to the student by building a GUI based modeling tool. This GUI based tool is very easy to use and provides a satisfactory modeling environment and includes some of the features used in the actual research environment. The students can develop innovative thermal management policies and can easily plug them to the existing tool so as better understand the thermal limits of present day manycore chip design.

# 7 RELATED PUBLICATIONS

1. Minki Cho, Nikhil Sathe, Arijit Raychowdhury, Saibal Mukhopadhyay, "Optimization of Burn-in Test for Many-core processors through Adaptive Spatiotemporal Power Migration", ITC 2010

2. Minki Cho, Nikhil Sathe, Man Prakash Gupta, Satish Kumar, Sudhakar Yalamanchilli and Saibal Mukhopadhyay, "Proactive Power Migration to Reduce Maximum Value and Spatiotemporal Non-uniformity of On-chip Temperature Distribution in Homogeneous Many-Core Processors", Semi-Therm 2010

# 8 APPENDIX

```
Function main ( ) {
        define and initialize local variables
        read configuration file
        read command line arguments
        read floorplan file
        populate R model using values specified in floorplan and configuration files
        populate C model if transient simulation is required
        setup initial temperatures
        while (! Power trace file empty) {
                power = read power values
                compute temperature for given sampling interval using "power". Store
                        temperature values into "temp".
                store "temp" into file
        }
        Dump steady state temperature into file
        Exit
}
```

**Figure 8-1: High level pseudo-code of HotSpot**

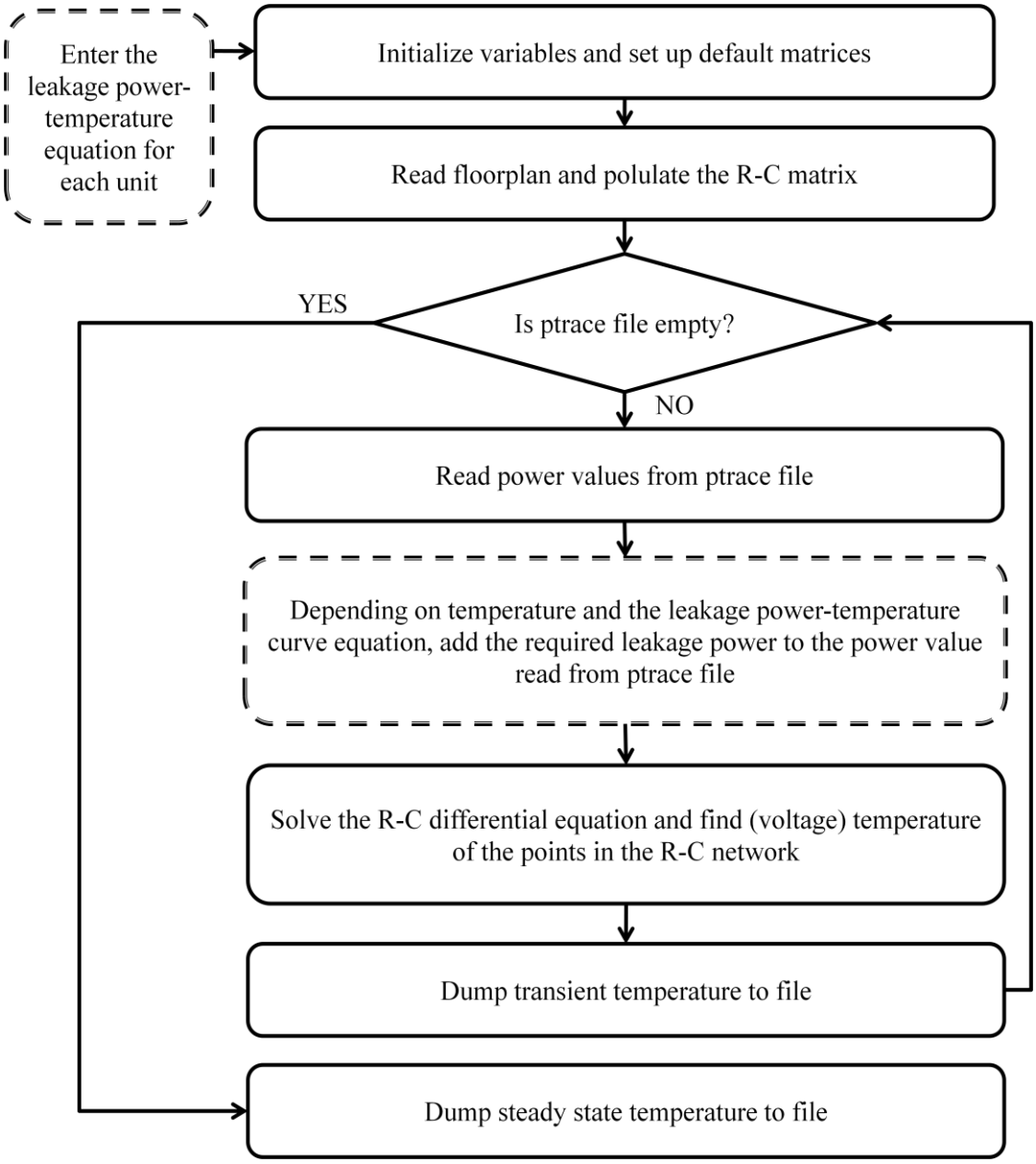



**Figure 8-2: Flow of HotSpot with leakage power – temperature interaction**

```
Function main ( ) {
        define and initialize local variables
        read configuration file
        read command line arguments
        read floorplan file
        populate R model using values specified in floorplan and configuration files
        populate C model if transient simulation is required
        setup initial temperatures
        read cache_leakage_coeff_file and core_leakage_coeff_file and store values in
        "leak_coeff" array

        while (! Power trace file empty) {
                power = read power values
                power = power + Leakage Power ( found using "temp" and "leak_coeff")
                compute temperature for given sampling interval using "power". Store
                        temperature values into "temp".

                store "temp" into file
        }
        Dump steady state temperature into file
}
```

**Figure 8-3: Implementation of leakage power - temperature interaction**

```
Preprocessing::
        Prepare core_leakage_coeff_file with just 1 leakage equation
        Prepare cache_leakage_coeff_file with just 1 leakage equation
```

```
Function main ( ) {
        define and initialize local variables
        read configuration file
        read command line arguments
        read floorplan file
        populate R model using values specified in floorplan and configuration files
        populate C model if transient simulation is required
        setup initial temperatures
        read cache_leakage_coeff_file and core_leakage_coeff_file and randomly store values in
        "leak_coeff" array. All the values in "leak_coeff" array will be same (but will be different for
        core blocks and cache blocks)

        while (! Power trace file empty) {
                power = read power values
                power = power + Leakage Power ( found using "temp" and "leak_coeff")
                compute temperature for given sampling interval using "power". Store temperature
                        values into "temp".

                store "temp" into file
        }
        Dump steady state temperature into file
}
```

**Figure 8-4: Pseudo-code implementing process variation**

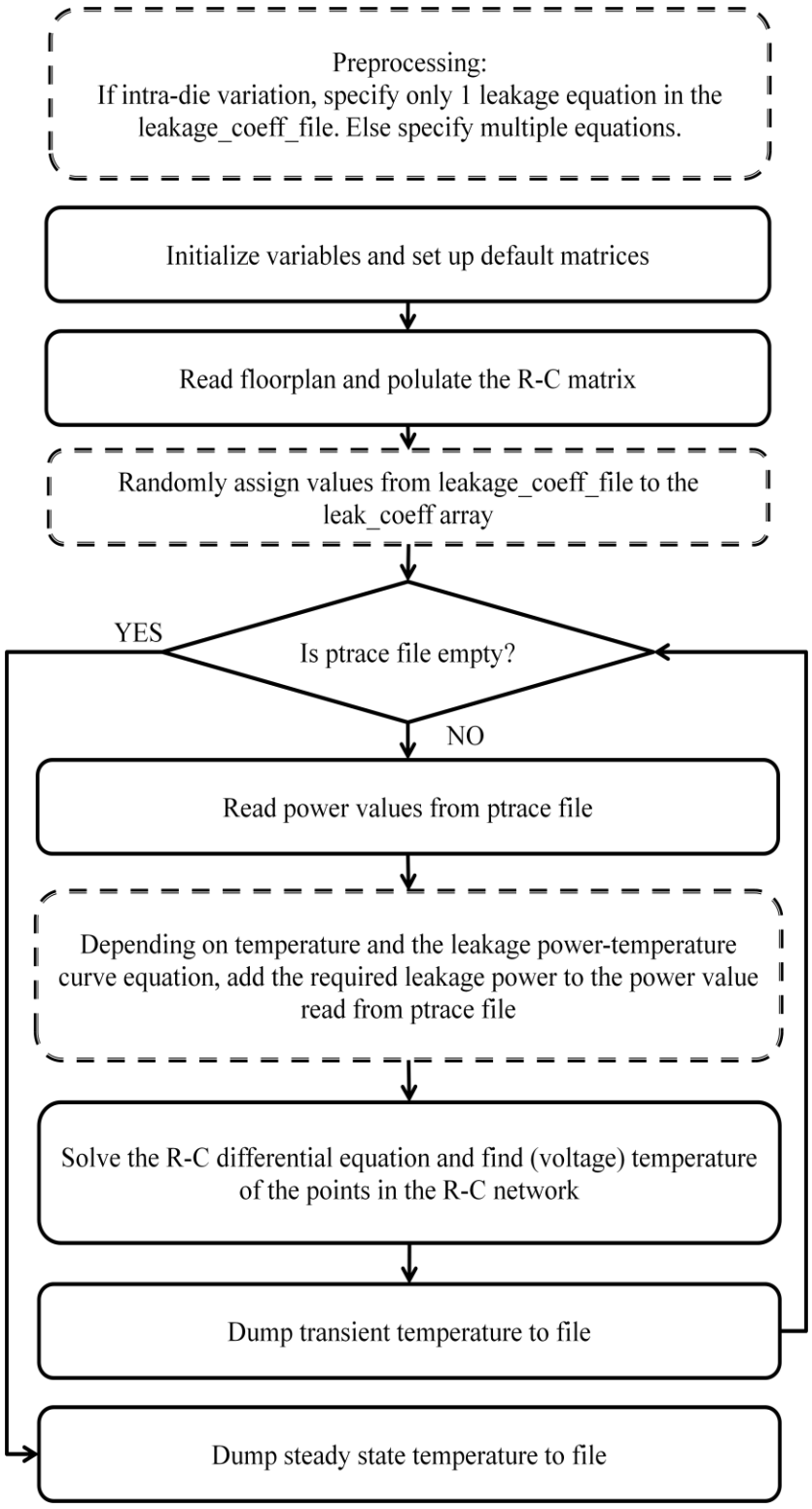



**Figure 8-5: Flow diagram of implementation of inter-die and intra-die process variation**

```
Function main ( ) {
        Initialize dead_cycles = Required dead cycle interval
        define and initialize local variables
        read configuration file, command line arguments, floorplan file
        populate R model using values specified in floorplan and configuration files
        populate C model if transient simulation is required
        setup initial temperatures
        power = read power trace file
        while (1) {
                if (migration)
                        Modify "power" array. Insert dead cycle power
                        Compute temperature for "dead_cycles" time interval.
                 else
                        power = power

                power = power + Leakage Power ( found using "temp" and "leak_coeff")
                if (migration) {
                        compute temperature for given (sampling interval – dead_cycles)
                } else {
                        compute temperature for given (sampling interval)
                }

                store "temp" into file
                time = time + time_step
                if(time > max_runtime) {
                        break
                }
        }
        Dump steady state temperature into file
}
```

**Figure 8-6: Pseudo-code showing implementation of dead cycles**

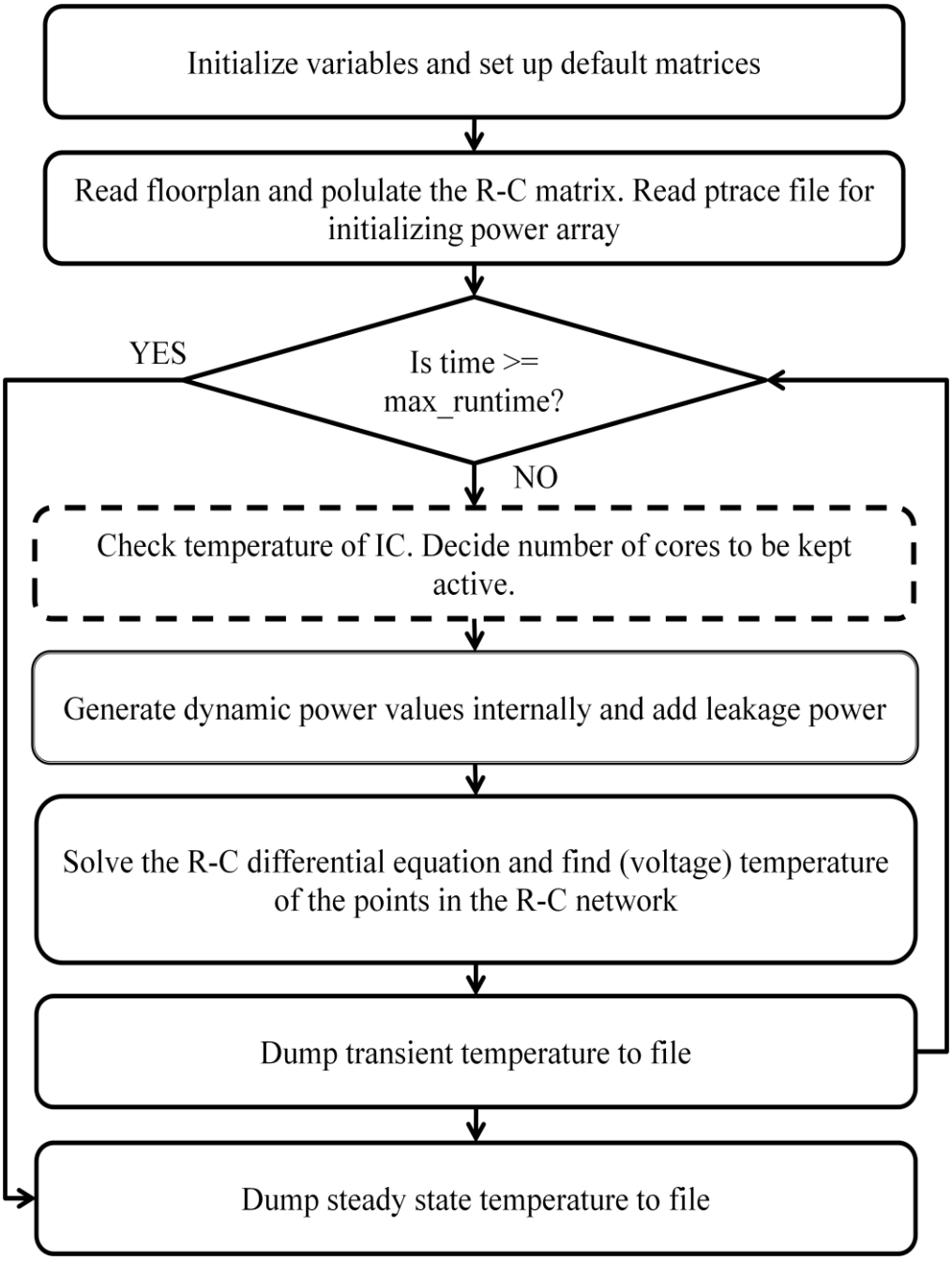



**Figure 8-7: Flow diagram: Implementing adaptation of number of active cores**

```
Function main ( ) {
        define and initialize local variables
        read configuration file
        read command line arguments
        read floorplan file
        populate R model using values specified in floorplan and configuration files
        populate C model if transient simulation is required
        setup initial temperatures
        power = read power trace file
        while (1) {
                if (sensor sample available) {
                        if (temperature crosses threshold)
                                reduce power. Modify "power" array
                         else
                                power = power
                }
                power = power + Leakage Power ( found using "temp" and "leak_coeff")
                compute temperature for given sampling interval using "power". Store temperature
                        values into "temp".
                store "temp" into file
                time = time + time_step
                if(time > max_runtime) {
                        break
                }
        }
        Dump steady state temperature into file
}
```

**Figure 8-8: Pseudo-code for implementing reactive policies**

```
Function main ( ) {
        define and initialize local variables
        read configuration file
        read command line arguments
        read floorplan file
        populate R model using values specified in floorplan and configuration files
        populate C model if transient simulation is required
                                        setup initial temperatures
        power = read power trace file
        while (1) {
                if (proactive migration interval)
                        Modify "power" array. Migrate cores.
                 else
                        power = power
                power = power + Leakage Power ( found using "temp" and "leak_coeff")
                compute temperature for given sampling interval using "power". Store temperature
                    values into "temp".
                store "temp" into file
                time = time + time_step
                if(time > max_runtime) {
                    break
                }
        Dump steady state temperature into file
}
```

**Figure 8-9: Pseudo-code for implementing proactive migration**

***Preprocessing:***
***Store the power trace of threads into files.***

```
Function main ( ) {
        Prepare core-thread mapping array. Randomly assign threads to cores.
        define and initialize local variables
        read configuration file, command line arguments, floorplan file
        populate R model using values specified in floorplan and configuration files
        populate C model if transient simulation is required
        setup initial temperatures
        power = read power trace file
        while (1) {
                if (Thread swapping) {
                        change core-thread mapping array
                }
                power = power + Leakage Power ( found using "temp" and "leak_coeff")
                compute temperature for given sampling interval

                store "temp" into file
                time = time + time_step
                if(time > max_runtime) {
                        break
                }
        }
        Dump steady state temperature into file
}
```

**Figure 8-10: Flow diagram showing implementation of thread swapping**

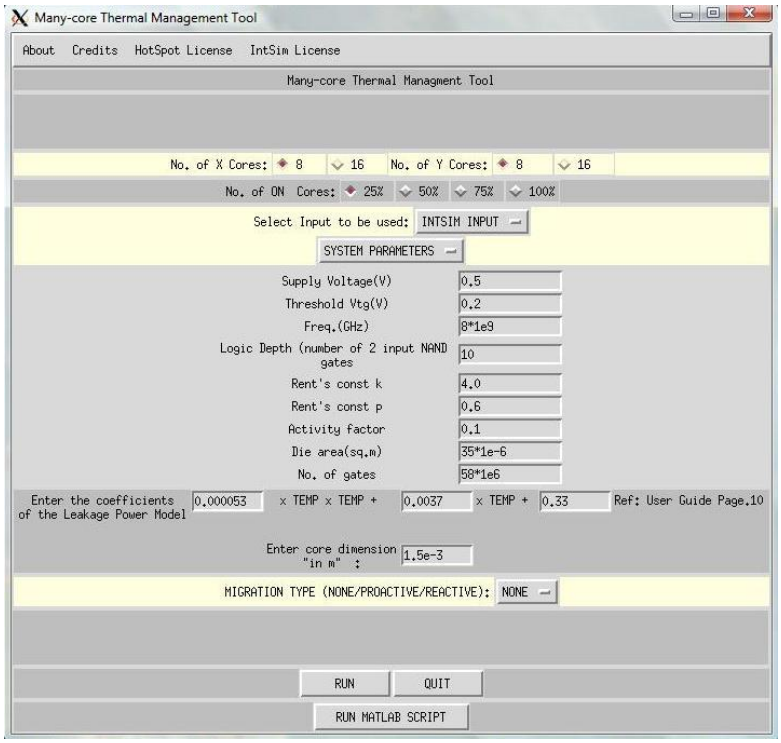



**Figure 8-11: IntSim design parameters**

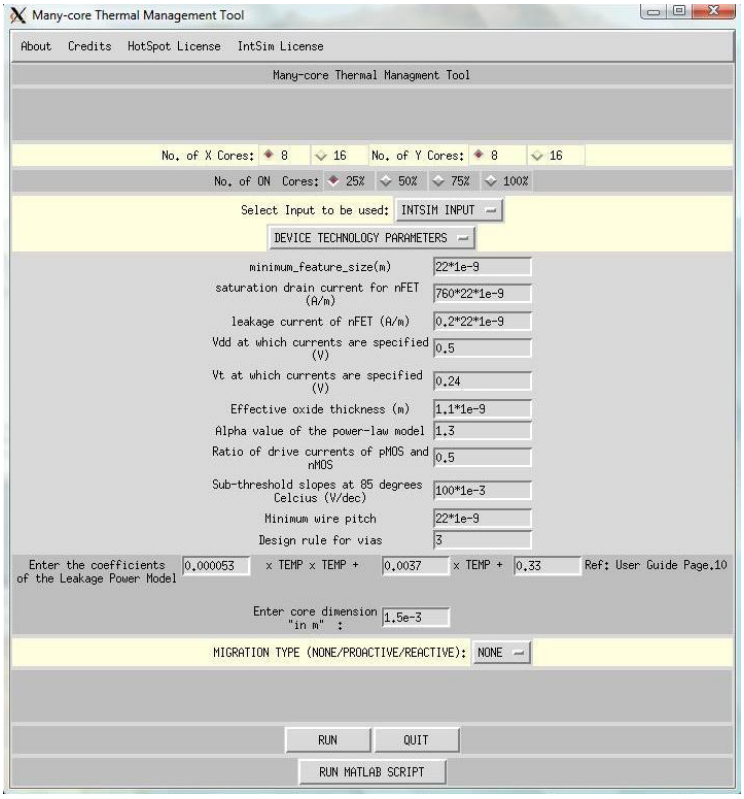



**Figure 8-12: IntSim device technology parameters**

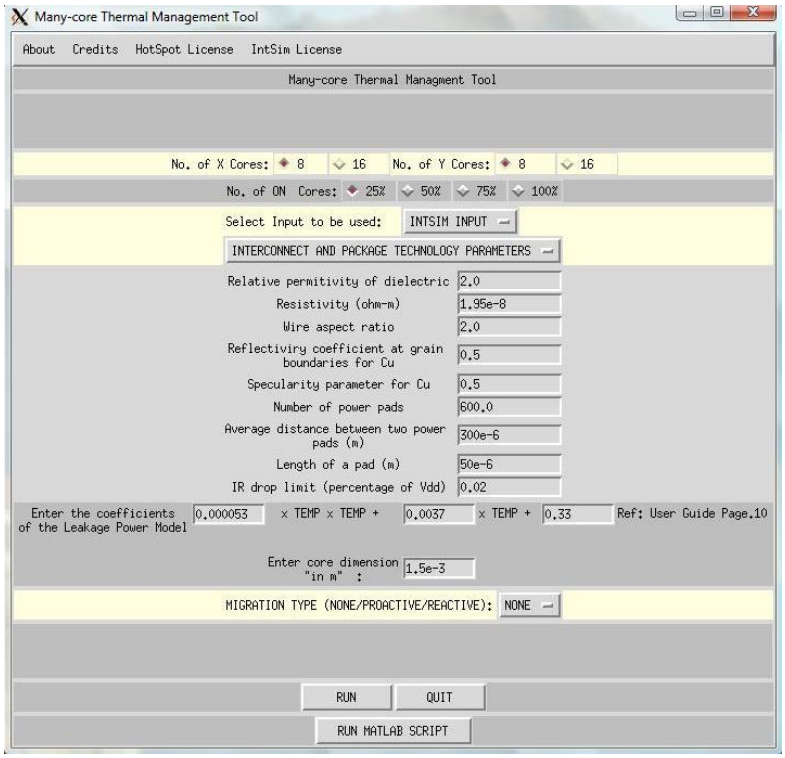



**Figure 8-13: IntSim interconnect and package technology parameters**

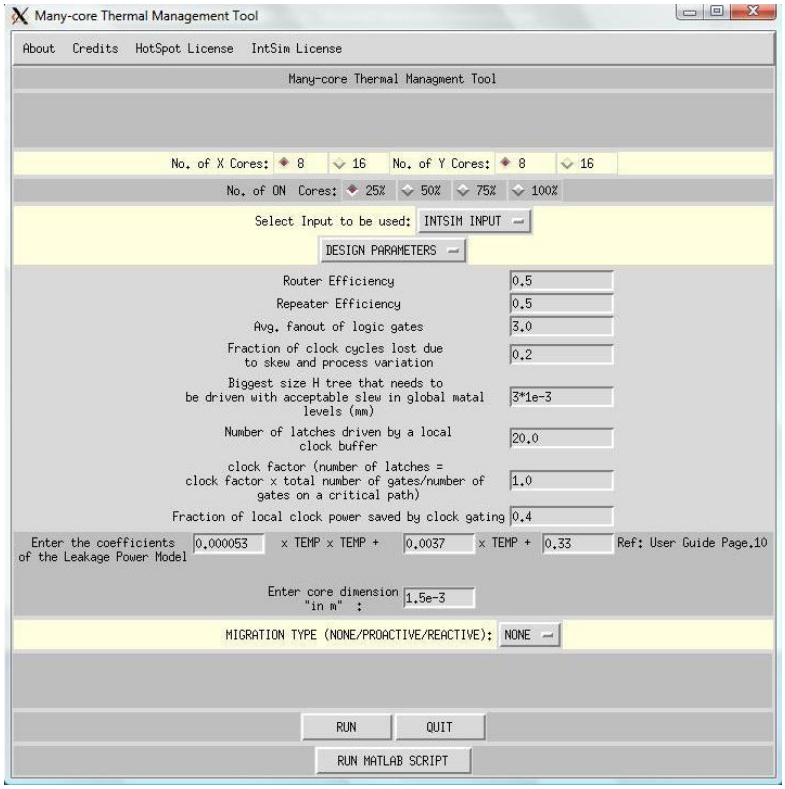



**Figure 8-14: IntSim design parameters**

# REFERENCES


[1] S. Borkar, “Thousand Core Chips—A Technology Perspective,” DAC 2007

[2] S. Mukhopadhyay et al., IEEE Trans. VLSI System, pp. 716-730, Aug. 2003

[3] M.Cho et al, “Proactive Power Migration to Reduce Maximum Value and Spatiotemporal Non-uniformity of On-chip Temperature Distribution in Homogeneous Many-Core Processors”, SemiTherm2010

[4] M. Cho et al, “Optimization of Burn-in Test for Many-core processors through Adaptive Spatiotemporal Power Migration”, ITC 2010

[5] K. Roy et. al. “Leakage current mechanisms and leakage reduction techniques in deep-submicron CMOS circuits”, *Proceedings of the IEEE*, vol. 91, No. 2, Feb, 2003.

[6] Tschanz et al, “ Dynamic sleep transistor and body bias for active leakage power control of microprocessors”, IEEE Journal of Solid State Circuits, November 2003.

[7] Vangal et al, “An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS”, ISSCC 2007.

[8] D. C. Sekar et. al, “Intsim: a CAD tool for optimization of multilevel interconnect networks”, IEEE/ACM ICCAD, 2007.

[9] International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: http://public.itrs.net/ (06/24/2010)

[10] Predictive Technology Model (PTM). [Online] Available: http://www.eas.asu.edu/~ptm/ (06/24/2010)

[11] W. Huang et. al, “Many-core design from a thermal perspective”, DAC 2008

[12] W. Huang et. al, “Hotspot: A Compact Thermal Modeling Method for CMOS VLSI systems”, IEEE TVLSI, pp. 501–513, May 2006

[13] A. Coskun et. al, “Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs”, IEEE TVLSI 2008

[14] D. Brooks, et. al, “Dynamic Thermal Management for High-Performance Microprocessors”, HPCA 2002

[15] P. Chaparro, et. al, “Understanding the Thermal Implications of Multicore Architectures”, IEEE Transaction on Parallel and Distributed Systems, 2007.

[16] R. McGowen, et. al, "Power and temperature control on a 90-nm Itanium family processor", IEEE JSSC, January 2006. pp. 29-237.

[17] J. Tschanz, et. al., "Adaptive Frequency and Biasing Techniques for Tolerance to Dynamic Temperature-Voltage Variations and Aging," ISSCC, 2007.

[18] J Srinivasan, et. al, "The Case for Lifetime Reliability-Aware Microprocessors," ISCA, 2004, pp. 276- 287.

[19] K. Sankaranarayanan, et., al, "A Case for Thermal-Aware Floorplanning at the Microarchitectrual Lelvel", Journal of Instruction-Level Parallelism, 2005.

[20] R. Rao et. al, "Throughput of multi-core processors under thermal constraints," ISLPED 2007

[21] E. Kursun et. al, "Temperature Variation Characterization and Thermal Management of Multicore Architectures," IEEE Micro 2009

[22] V. Gektin, et. al, "Substantiation of Numerical Analysis Methodology for CPU Package with Non-uniform Heat Dissipation and Heat Sink with Simplified Fin Modeling", Thermal and Thermomechanical Phenomena in Electronic Systems, 2004.

[23] S. Bell, et. al., "TILE64 Processor: A 64-Core SoC with Mesh Interconnect," IEEE ISSCC, 2008, pp. 88-598.

[24] S. R. Vangal, et. al., "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS", *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, Jan. 2008, pp. 29-41.

[25] D. C. Sekar et. al. "Optimal Signal, Power, Clock and Thermal Interconnect Networks for High-Performance 2D and 3D Integrated Circuits", PhD Dissertation, Georgia Institute of Technology, 2008

[26] http://www.intel.com/products/processor_number/chart/xeon.htm (06/24/2010)

[27] http://lava.cs.virginia.edu/HotSpot/index.htm (06/24/2010)

[28] Tiwari et al,"Dynamic Power Management for Microprocessors:A Case Study", VLSI'97

[29] [Online].http://www.intel.com/pressroom/kits/events/moores_law_40th/index.htm?iid=tech_mooreslaw+body_presskit (06/24/2010)

[30] Krum, Thermal management, in: F. Kreith (Ed.), The CRC handbook of thermal engineering, CRC Press, Boca Raton, FL, 2000, pp. 2.1-2.92.

[31] [Online].http://www.intel.com/pressroom/kits/events/moores_law_40th/index.htm?iid=tech_mooreslaw+body_presskit (06/24/2010)

[32] T. Pering, T. Burd, and R. Brodersen," The simulation and evaluation of dynamic voltage scaling algorithms", ISLED August 1998

[33] Advanced Configuration and Power Interface.http://www.teleport.com/ acpi/.chip

[34] L2 cache controller. *Digest* of *Technical Papers - IEEEInternational Solid-State Circuits Conference,* 40:412, 1997.

[35] E. Rohou and M. Smith. "Dynamically managing processor temperature and power", in *2nd Workshop on Feedback-Directed Optimization,* Nov. 1999.

[36] H. Sanchez et al., "Thermal management system for high performance power-pc microprocessors", *Digest of Papers -COMPCON - IEEE Computer Society International Conference,*page 325, 1997.

[37] JEDEC Solid State Technology Association, Arlington, VA, "Failure mechanisms and models for semiconductor devices," JEDEC publication JEP122C, 2006.

[38] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur, "Thermal performance challenges from silicon to systems," *Intel Technol. J., Q3*, vol. 23, p. 16, 2000.

[39] Y. Li, B.C. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP Design Space Exploration Subject to Physical Constraints," HPCA, Feb. 2006.

[40] Y. Li, K. Skadron, Z. Hu, and D. Brooks, "Performance, Energy, and Thermal Considerations for SMT and CMP Architectures," HPCA, pp. 71-82, Feb. 2005.

[41] Powell et al. "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System"

[42] Fu Zhaoguo et al, "A task scheduling algorithm of real-time leakage power and temperature optimization for MPSoC", 11th IEEE International Conference on Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09.

[43] Chee How Lim, Robert Daash, and George Cai. "A Thermal-Aware Superscalar Microarchitecture", ISQED pages 517–522, San Jose, California, USA, March 2002.

[44] Michael Kanellos. "At Intel - the chip with two brains". *C—Net news.com*, Aug 2002.

[45] Coskun et al, "Temperature Aware Task Scheduling in MPSoCs", DATE'07

[46] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A.Weger, P. Bose, “Thermalaware task scheduling at the system software level,” ISLPED, pp. 213-218, 2007.

[47] A. Kumar, L. Shang, L.-S. Peh, N. Jha, “HybDTM: A coordinated hardware-software approach for dynamic thermal management,” DAC,pp. 548-553, 2006.

[48] Inchoon Yeo; Chih Chun Liu; Eun Jung Kim, “Predictive Dynamic Thermal Management for Multicore Systems”, DAC 2008

[49] Intel Developer Forum, 2004

[50] K. A. Bowman et al., "Impact of Die-to-Die and Within-Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration", IEE JSSC, 2002, pp. 183-190.

[51] S. Borkar et al., "Parameter Variations and Impact on Circuits and Micro-architetture", DAC, 2003, pp. 338-342.

[52] Stan et al, “HotSpot: a Dynamic Compact Thermal Model at the Processor-Architecture Level”, Therminic 2002

[53] W. Huang, K. Skadron, S. Gurumurthi, R. J. Ribando, and M. R. Stan. " Differentiating the Roles of IR Measurement and Simulation for Power and Temperature-Aware Design." In Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Apr. 2009

[54] https://opensource.ncsu.edu/FreePDK (06/24/2010)

[55] Loh et al, “Zesto: A cycle-level simulator for highly detailed microarchitecture exploration”, ISPASS 2009

[56] Developed by CASL, Georgia Institute of Technology
http://www.ece.gatech.edu/research/labs/casl/index.html (06/24/2010)